Lecture Notes in Computer Science 1874
Edited by G. Goos, J. Hartmanis and J. van Leeuwen

Springer
Berlin
Heidelberg
New York
Barcelona
Hong Kong
London
Milan
Paris
Singapore
Tokyo

Yahiko Kambayashi Mukesh Mohania
A Min Tjoa (Eds.)

Data Warehousing and Knowledge Discovery

Second International Conference, DaWaK 2000
London, UK, September 4-6, 2000
Proceedings

Springer

Series Editors

Gerhard Goos, Karlsruhe University, Germany
Juris Hartmanis, Cornell University, NY, USA
Jan van Leeuwen, Utrecht University, The Netherlands

Volume Editors

Yahiko Kambayashi
Kyoto University, Graduate School of Informatics
Yoshida-Honmachi, Sakyo-ku, Kyoto 606-8501, Japan
E-mail: yahiko@kuis.kyoto-u.ac.jp

Mukesh Mohania
Western Michigan University, Computer Science Department
Kalamazoo, MI 49008, USA
E-mail: mohania@cs.wmich.edu

A Min Tjoa
Vienna University of Technology, IFS
Favoritenstr. 9-11/188, 1040 Vienna, Austria

Cataloging-in-Publication Data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Data warehousing and knowledge discovery : second international conference ; proceedings / DaWaK 2000, London, UK, September 4 - 6, 2000. Yahiko Kambayashi ... (ed.). - Berlin ; Heidelberg ; New York ; Barcelona ; Hong Kong ; London ; Milan ; Paris ; Singapore ; Tokyo : Springer, 2000
(Lecture notes in computer science ; Vol. 1874)
ISBN 3-540-67980-4

CR Subject Classification (1998): H.2, H.3, C.2, H.4, H.5

ISSN 0302-9743
ISBN 3-540-67980-4 Springer-Verlag Berlin Heidelberg New York

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer-Verlag. Violations are liable for prosecution under the German Copyright Law.

Springer-Verlag Berlin Heidelberg New York
a member of BertelsmannSpringer Science+Business Media GmbH
© Springer-Verlag Berlin Heidelberg 2000
Printed in Germany

Typesetting: Camera-ready by author
Printed on acid-free paper SPIN 10722353 06/3142 5 4 3 2 1 0

Preface

The Second International Conference on Data Warehousing and Knowledge Discovery (DaWaK 2000) was held in Greenwich, UK 4–6 September. DaWaK 2000 was a forum where researchers from data warehousing and knowledge discovery disciplines could exchange ideas on improving next generation decision support and data mining systems.

The conference focused on the logical and physical design of data warehousing and knowledge discovery systems. The scope of the papers covered the most recent and relevant topics in the areas of data warehousing, multidimensional databases, OLAP, knowledge discovery and mining complex databases. These proceedings contain the technical papers selected for presentation at the conference.

We received more than 90 papers from over 20 countries and the program committee finally selected 31 long papers and 11 short papers. The conference program included three invited talks, namely, "A Foolish Consistency: Technical Challenges in Consistency Management" by Professor Anthony Finkelstein, University College London, UK; "European Plan for Research in Data Warehousing and Knowledge Discovery" by Dr. Harald Sonnberger (Head of Unit A4, Eurostat, European Commission); and "Security in Data Warehousing" by Professor Bharat Bhargava, Purdue University, USA.

We would like to thank the DEXA 2000 workshop general chair (Professor Roland Wagner) and the organizing committee of the 11th International Conference on Database and Expert Systems Applications (DEXA 2000) for their support and cooperation. Many many thanks are due Ms Gabriela Wagner for providing a great deal of help and assistance. We are very indebted to all program committee members and outside reviewers who have very carefully and timely reviewed the papers. We would also like to thanks all the authors who submitted their papers to DaWaK 2000; they provided us with an excellent technical program.

Yahiko Kambayashi, General Chair
Mukesh Mohania and A Min Tjoa, Program Committee Chairs
Tok Wang Ling, Panel Chair

September 2000

Program Committee

General Chair:
Y. Kambayashi (Kyoto University Sakyo, Japan)

Program Chairs:
M. Mohania (University of South Australia)
A M. Tjoa (Vienna University of Technology, Austria)

Panel Chair:
T.W. Ling (National University of Singapore)

Program Committee:
D. Agrawal (University California Santa Barbara)
P. Atzeni (Università di Roma Tre, Italy)
E. Baralis (Politecnico di Torino, Italy)
S.A. Becker (Florida Institute of Technology, USA)
S. Berchtold (stb Software Technologie Beratung GmbH, Germany)
B. Bhargava (Purdue University, USA)
T. Catarci (Università degli Studi di Roma "La Sapienza", Italy)
S. Chakravarthy (University of Florida, USA)
P. Chamoni (University Duisburg, Germany)
Q. Chen (HP Labs, USA)
Arbee L.P. Chen (National Tsing-Hua University, Taiwan)
C.-M. Chen (Telcordia Technologies, USA)
S. Choenni (National Aerospace Laboratory, The Netherlands)
P.K. Chrysanthis (University of Pittsburgh, USA)
G. Dong (Wright State University, USA)
V. Estivill-Castro (The University of Newcastle, Australia)
U. Fayyad (Microsoft, USA)
S. Goel (Informix, USA)
A. Gupta (Western Michigan University, USA)
S. K. Gupta (Indian Institute of Technology Delhi, India)
M. Gyssens (University of Limburg, Belgium)
J. Han (Simon Fraser University, Canada)
J. Harrison (University of Queensland, Australia)
K. Karlapalem (Hong Kong University of Science and Technology, China)
S. Kaski (Helsinki University of Technology, Finland)
H. Kawano (Kyoto University, Japan)
L. Kerschberg (George Mason University, USA)
M. Kitsuregawa (University of Tokyo, Japan)
F. Korn (AT&T, USA)
V. Kumar (University of Missouri-Kansas City, USA)
L. V.S. Lakshmanan (Concordia University, Canada)
D. Lee (Hong Kong University of Science and Technology, China)
W. Lehner (IBM Almaden Research Center, USA)
L. Libkin (Bell Labs, USA)

Q. Li (City University of Hong Kong, China)
T.W. Ling (National University of Singapore)
H. Lu (Hong Kong University of Science and Technology, China)
S. Madria (Purdue University, USA)
H. Mannila (Microsoft, USA)
S. Navathe (Georgia Institute of Technology, USA)
R. Ng (University of British Columbia, Canada)
W.-K. Ng (Nanyang Technological University, Singapore)
S. Nishio (Osaka University, Japan)
S. Paraboschi (Politecnico di Milano, Italy)
D.S. Parker (University of California, USA)
S. Pramanik (Michigan State University, USA)
C. Rainsford (Defence Science Technology Organisation, Australia)
J. Roddick (University of South Australia)
E. A. Rundensteiner (Worcester Polytechnic Institute, USA)
D. Sacca (Università della Calabria, Italy)
S. Samtani (Telcordia Technologies, USA)
N.L. Sarda (Indian Institute of Technology Mumbai, India)
M. Schrefl (University of South Australia, Australia)
T. Sellis (National Technical University of Athens, Greece)
K. Shim (Bell Labs, USA)
I.-Y. Song (Drexel University, USA)
J. Srivastava (University of Minnesota, USA)
E. Teniente (Universitat Politecnica de Catalunya, Spain)
H. Toivonen (Nokia Research Center, Finland)
M. Vincent (University of South Australia)
B. Wuthrich (Hong Kong University of Science and Technology, China)
J. Zytkow (University of North Carolina, USA)

Table of Contents

Data Warehouse Design

The Design and Development of a Logical System for OLAP 1
Cabibbo L., Torlone R.; Italy

Applying Vertical Fragmentation Techniques in Logical Design of 11
Multidimensional Databases
Golfarelli M., Maio D., Rizzi S.; Italy

Space-Efficient Data Cubes for Dynamic Environments 24
Riedewald M., Agrawal D., El Abbadi A., Pajarola R.; USA

On Making Data Warehouses Active 34
Schrefl M., Thalhammer T.; Australia, Austria

Materialized Views

Supporting Hot Spots with Materialized Views 47
Albrecht J., Bauer A., Redert M.; Germany

Evaluation of Materialized View Indexing in Data Warehousing Environments 57
Bellatreche L., Karlapalem K., Li Q.; China

View Derivation Graph with Edge Fitting for Adaptive Data Warehousing 67
Stanoi I., Agrawal D., El Abbadi A.; USA

On the Importance of Tuning in Incremental View Maintenance:
An Experience Case Study 77
O'Gorman K., Agrawal D., El Abbadi A.; USA

Warehouse Data Creation and Maintenance

BEDAWA - A Tool for Generating Sample Data for Data Warehouses 83
Huynh T.N., Nguyen B.T., Schiefer J., Tjoa A M.; Austria

DyDa: Dynamic Data Warehouse Maintenance in a Fully Concurrent
Environment 94
Zhang X., Rundensteiner E.A.; USA

Scalable Maintenance of Multiple Interrelated Data Warehousing Systems 104
Ding L., Zhang X., Rundensteiner E.A.; USA

View Maintenance for Hierarchical Semistructured Data 114
Liefke H., Davidson S.B.; USA

Maintaining Horizontally Partitioned Warehouse Views 126
Xu M., Ezeife C.I.; Canada

Invited Talk:

Funding Research in Data Warehousing and Knowledge Discovery
EPROS: The European Plan for Research in Official Statistics 134
Mercy J.-L., Sonnberger H.; Luxembourg

Warehouse Views Selection and Evolution

Elimination of Redundant Views in Multidimensional Aggregates 146
Kotsis N., McGregor D.R.; United Kingdom

Data Cube Compression with QuantiCubes 162
Furtado P., Madeira H.; Portugal

History-Driven View Synchronization 168
Koeller A., Rundensteiner E.A.; USA

A Logical Model for Data Warehouse Design and Evolution 178
Bouzeghoub M., Kedad Z.; France

OLAP System Design and Query Analysis

An Alternative Relational OLAP Modeling Approach 189
Bauer A., Hümmer W., Lehner W.; Germany

Functional Dependencies in Controlling Sparsity of OLAP Cubes..... 199
Niemi T., Nummenmaa J., Thanisch P.; Finland

An OLAP-based Scalable Web Access Analysis Engine..... 210
Chen Q., Dayal U., Hsu M.; USA

PROMISE: Predicting Query Behavior to Enable Predictive Caching
Strategies for OLAP Systems 224
Sapia C.; Germany

OLAP Query Evaluation

Supporting Online Queries in ROLAP 234
Barbará D., Wu X.; USA

Optimal Multidimensional Query Processing Using Tree Striping 244
Berchtold S., Böhm C., Keim D.A., Kriegel H.-P., Xu X.; Germany

Enhancing Preprocessing in Data-Intensive Domains using Online-Analytical Processing..... 258
Maedche A., Hotho A., Wiese M.; Germany

Meta-queries - Computation and Evaluation 265
Ben-Eliyahu-Zohary R., Gudes E.; Israel

Partitioning Algorithms for the Computation of Average Iceberg Queries 276
Bae J., Lee S.; Korea

Invited Talk:
Security in Data Warehousing..... 287
Bhargava B.; USA

Association Rules

Mining of Association Rules in Text Databases Using Inverted Hashing and Pruning 290
Holt J.D., Chung S.M.; USA

SQL Based Association Rule Mining Using Commercial RDBMS (IBM DB2 UBD EEE) 301
Yoshizawa T., Pramudiono I., Kitsuregawa M.; Japan

On Supporting Interactive Association Rule Mining 307
Goethals B., Van den Bussche J.; Belgium

Temporal Association Rules

Discovering Temporal Patterns for Interval-Based Events 317
Kam P.-s., Fu A.W.-c.; China

An Integrated Query and Mining System for Temporal Association Rules 327
Chen X., Petrounias I. ; United Kingdom

Mining Changes for Real-Life Applications 337
Liu B., Hsu W., Han H.-S., Xia Y.; Singapore

AIM: Approximate Intelligent Matching for Time Series Data 347
Kim E.D., Lam J.M.W., Han J.; Canada

Mining Complex Databases

COFE: A Scalable Method for Feature Extraction from Complex Objects 358
Hristescu G., Farach-Colton M.; USA

The Pruning Power: Theory and Heuristics for Mining Databases with Multiple *k*-Nearest-Neighbor Queries 372
Böhm C., Braunmüller B., Kriegel H.-P.; Germany

Data Mining Support in Database Management Systems 382
Morzy T., Wojciechowski M., Zakrzewicz M.; Poland

Decision Trees for Probabilistic Data 393
Aboa J.-P., Emilion R.; France

Mining Frequent Binary Expressions 399
Calders T., Paredaens J.; Belgium

A Fast Algorithm for Hierarchical Text Classification 409
Chuang W.T., Tiyyagura A., Yang J., Giuffrida G.; USA

A Hybrid Technique for Data Mining on Balance-Sheet Data 419
Dattilo G., Greco S., Masciari E., Pontieri L.; Italy

Mondou: Information Navigator with Visual Interface 425
Kawano H., Kawahara M.; Japan

Vmhist: Efficient Multidimensional Histograms with Improved Accuracy 431
Furtado P., Madeira H.; Portugal

Author Index 437

The Design and Development of a Logical System for OLAP*

Luca Cabibbo and Riccardo Torlone

Dipartimento di Informatica e Automazione, Università di Roma Tre
Via della Vasca Navale, 79 — I-00146 Roma, Italy
{cabibbo,torlone}@dia.uniroma3.it

Abstract. We report on the design and development of a novel architecture for data warehousing. This architecture adds a "logical" level of abstraction to the traditional data warehousing framework, to guarantee an independence of OLAP applications from the physical storage structure of the data warehouse. This property allows users and applications to manipulate multidimensional data ignoring implementation details. Also, it supports the integration of multidimensional data stored in heterogeneous OLAP servers. We propose $\mathcal{MD}$, a simple data model for multidimensional databases, as the reference for the logical layer. We then describe the design of a system, called MDS, that supports the above logical architecture.

1 Introduction

Nowadays, current technology provides a lot of software tools supporting data warehousing [3–6]. Apart a number of facilities implementing specific activities, the main components of a data warehousing system are the *data warehouse server* and the front-end clients, often called *OLAP tools*. Data warehouse servers can be relational (ROLAP), multidimensional (MOLAP), or hybrid systems. ROLAP systems store a data warehouse in relational tables and maps multidimensional operations to extended SQL statements. A quite standard organization of a data warehouse in a ROLAP system is the *star scheme* (or variant thereof) [5], with a central table representing the fact on which the analysis is focused, and a number of tables, only partially normalized, representing the dimensions of analysis (e.g., time, location, type). Conversely, MOLAP systems are special servers that directly store and manipulate data in multidimensional proprietary formats. In hybrid OLAP systems data can be stored in both relational and multidimensional form. Front-end tools offer powerful querying and reporting capabilities, usually based on interactive graphical user interfaces similar to spreadsheets. Standard Application Program Interfaces (API) for communication between data warehouse server and clients are also emerging (e.g., OLE DB for OLAP [7] and MD-API [6]).

Although the available tools are powerful and very useful, we believe that there is a limitation in the traditional organization of a data warehousing system.

* This work was partially supported by *IASI*, by *CNR* and by *MURST*.

Y. Kambayashi, M. Mohania, and A M. Tjoa (Eds.): DaWaK 2000, LNCS 1874, pp. 1–10, 2000.
© Springer-Verlag Berlin Heidelberg 2000

The problem is that the way in which information is viewed and manipulated in an OLAP tool strictly depends on how data is stored in the server. Several OLAP tools somehow filter the underlying technology, by providing a high-level interface, but they are usually customized to work with a specific data warehouse server or class of servers (relational or multidimensional). Also and more important, if the organization of the data warehouse changes, there is often the need to rebuild the front-end views of the data warehouse. For example, if an OLAP tool provides a multidimensional view of a relational star scheme in the form of a spreadsheet and the scheme is just normalized (or denormalized) for efficiency reasons, the view is no more valid and needs to be redefined. Another consequence of the dependency of OLAP applications from their implementation is that the integration of data coming from different data warehousing systems is difficult to achieve, since no standardization exists in this context.

The goal of this paper is to illustrate a new architecture for data warehousing aimed at solving this problem. The basic idea is to add an explicit level of abstraction to the traditional data warehousing framework. The new level, which we call "logical", serves to guarantee an *independence* of OLAP applications from the physical storage structure of the data warehouse. This is similar to what happens with relational technology, in which the property of data independence allows users and applications to manipulate tables and views ignoring implementation details.

We make use of a simple data model, called $\mathcal{MD}$, as the reference for the logical level. This model provides a number of constructs used to describe, in an abstract but natural way, the basic notions that can be found in almost every OLAP system (fact, dimension, level of aggregation, and measure).

We then describe the design of a system, called MDS, that supports the above logical architecture. MDS is able to view an existing data warehouse, which can be stored in a variety of systems, in terms of the $\mathcal{MD}$ model and to map queries and data between the two frameworks. By providing a practical example of use of MDS, we show how the architecture we propose can make a data warehouse more effective and easier to use.

The rest of the paper is organized as follows. In Sect. 2 we give a general overview of the approach. Sect. 3 is devoted to the presentation of the system we have designed and, in Sect. 4, the most important component of this system is illustrated in detail with reference to a specific but important case.

2 A Logical Approach to Data Warehousing

According to Inmon [4], the traditional architecture for data warehousing comprises four different levels, as follows:

- *Operational layer.* The operational sources of data, which are often heterogeneous and distributed, from which the data warehouse is populated by means of cleaning, filtering, and integration operations.
- *Data Warehouse layer.* The data warehouse lies at this level and is kept separate from the operational sources. Data is represented here either in

relational form (ROLAP systems) or in multidimensional proprietary form (MOLAP systems).
- *Data mart layer.* Each data mart is generally built from a selected portion of the data warehouse for studying a specific, well-bounded business problem.
- *Individual layer.* Data is presented to analysts for interpretation. Various types of graphical representations are used here.

Variants of this organization are possible. In particular, either the data warehouse or the data mart layer can be absent. However, there is always an intermediate level of persistent data between the operational and the individual layers; we will refer to this level as to the data warehouse level hereinafter.

As we have said in the Introduction, a font of anomalies in the traditional architecture is that the way in which data is viewed and manipulated at the individual level strictly depends on the data warehouse level. As a consequence, changes to the organization of the data warehouse are propagated to the views defined on it, and thus to the external level. Moreover, the interoperability between different data warehouses must be solved case by case.

A possible solution to alleviate this problem is the introduction of a "logical" level in the data warehousing architecture. The new level serves to hide details of the actual implementation of the data warehouse and to concentrate on describing, in abstract terms, the basic, multidimensional aspects of data employed in their analysis. Specifically, the data warehouse level is split into a *logical layer*, which describes the content of the data warehouse in multidimensional but abstract terms, and an *internal layer*, which describes how the data warehouse is implemented (relational or multidimensional structures and their organization). Queries and views are defined with respect to the logical data warehouse layer. According to this organization, data warehouse restructuring occurs at the internal level and just requires the modification of the mapping between the internal and the logical layer. The logical representation of the data warehouse remains always the same and so queries and views defined on it.

We believe that this architecture has a number of further advantages: (i) analysts can manipulate data easier since they can refer to a high-level representation of data; (ii) interoperability between different data warehousing systems is facilitated, since the new level provides an abstract, and thus unifying, description of data; (iii) the system can scale-up easily, for instance in case of addition of new data sources, since this corresponds to a simple extension of the data warehouse description at the logical level.

The $\mathcal{MD}$ Data Model and Algebra. The MultiDimensional data model [1] ($\mathcal{MD}$ for short) is based on a few constructs modeling the basic concepts that can be found in any OLAP system. An $\mathcal{MD}$ *dimension* is a collection of *levels* and corresponds to a business perspective under which the analysis can be performed. The levels of a dimension are data domains at different granularity and are organized into a hierarchy. Within a dimension, values of different levels are related through a family of *roll-up functions*, according to the hierarchy defined on them. A roll-up function $\text{R-UP}_l^{l'}$ (also denoted by $\xrightarrow{\rho}$) associates a value v of

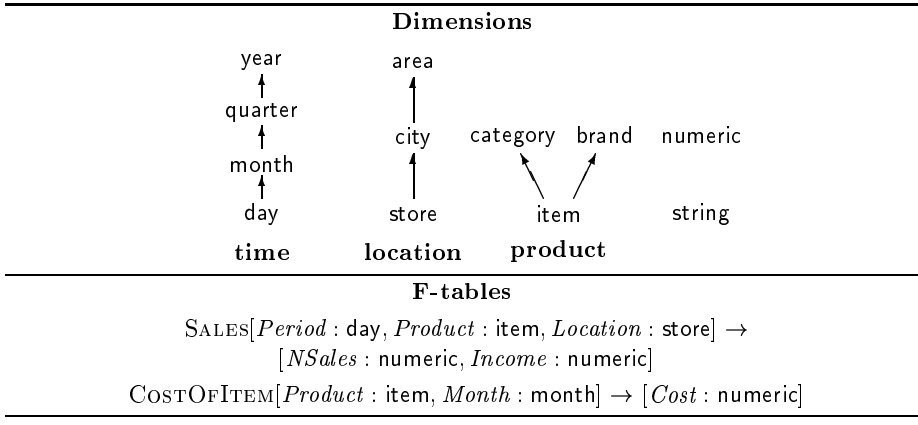


Fig. 1. The sample $\mathcal{MD}$ scheme `Retail`

a level l with a value v' of an upper level l' in the hierarchy. In this case we say that v *rolls up to* v' and also that l *rolls up to* l'. The main construct of the $\mathcal{MD}$ model is the *f-table*: this is a (partial) function that associates a collection of level values, called *symbolic coordinate*, with one or more *measures*. Components of coordinates are also called *attributes*. An *entry* of an f-table f is a coordinate over which f is defined. Thus, an f-table is used to represent factual data on which the analysis is focused (the measures) and the perspective of analysis (the coordinate).

As an example, consider the $\mathcal{MD}$ scheme `Retail`, shown in Fig. 1. This scheme can be used by a marketing analyst of a chain of toy stores and is organized along dimensions **time**, **product**, and **location** (shown on top of the figure). The **time** dimension is organized in a hierarchy of levels involving day, month, quarter, and year. Similarly, the **location** dimension is based on a hierarchy of levels involving store, city, and area. Finally, the **product** dimension contains levels item, category, and brand. There are two further *atomic* dimensions (that is, having just one level) that are used to represent **numeric** values and **string**s. In this framework, we can define the f-tables SALES and COSTOFITEM. The former describes summary data for the sales of the chain, organized along dimensions **time** (at day level), **product** (at item level), and **location** (at store level). The measures for this f-table are *NSales* (the number of items sold) and *Income* (the gross income), both having type numeric. The f-table COSTOFITEM is used to represents the costs of the various items, assuming that costs may vary from month-to-month.

In [2] we have defined a number of query languages for $\mathcal{MD}$ according to different needs: (i) a graphical and easy-to-use language for end-users; (ii) a declarative and powerful language (based on calculus) for experts; and (iii) a procedural language (based on an algebra) suitable to perform optimization and directly implementable. We now briefly recall the algebra for $\mathcal{MD}$ since it is used by the system as an intermediate language whose operators are directly

implemented or are translated into statements of a language (e.g., SQL or API) for an existing database server (a relational, a ROLAP, or a MOLAP system).

The $\mathcal{MD}$ algebra [2], apart from a number of operators that are variants of relational algebra operators, comprises three new operators. Two of them allow the application of scalar (φ) and aggregative functions (ψ) on attributes of an f-table. The other operator (ϱ) is specific for the $\mathcal{MD}$ model: it allows the application of a roll-up function to an f-table.

As an example, assume that we need to construct from the `Retail` database (Fig. 1) the f-table WEST-PROFIT storing, for each store and month, the net revenue of the stores in the west area. The net revenue is to be calculated as the difference between total income and cost of the items sold.

This query can be specified by means of the following $\mathcal{MD}$ algebra expression.

$$\psi^{netRevenue=sum(dr)}_{Product,Month}(\varphi^{dr=income-nsales*cost}(\text{COSTOFITEM} \bowtie_{Product,\ Month} \varrho^{month}_{period:day}(\sigma_{Area=West}(\varrho^{area}_{location:store}(\text{SALES})))))$$

The inner roll-up operator ϱ in the expression extends the coordinates of the f-table SALES with a new attribute *area* corresponding to the *store* of each sale. The σ operator is then used to select only the stores in the west area. The outer operator ϱ further extends the coordinates of the result with a new attribute *month* corresponding to the *day* of each sale. The equi-join operator $\bowtie$ is then used to combine the f-table so obtained and the f-table COSTOFITEM, according to the values over the attributes *Product* and *Month* (we recall that in our example the costs of items are stored on monthly basis). The operator φ introduces to the result f-table the measure *dr* (the daily revenue) computed from the other measures by the specified scalar function. Finally, the aggregate operator ψ computes the sum of the measure *dr* of the f-table we have obtained, grouping the result by the attributes *Product* and *Month*.

We have shown that the $\mathcal{MD}$ algebra is equivalent to $\mathcal{MD}$ calculus and more powerful than the $\mathcal{MD}$ graphical language. It is therefore possible to automatically translate graphical and textual queries into algebraic expressions.

3 Design of the System Architecture

On the basis of the issues discussed in previous sections, we have designed a tool for data warehousing, called MDS (MultiDimensional System), whose overall architecture is shown in Fig. 2. (In the figure, a box represents a software module, a cylinder represents a repository of data, and a line denotes a flow of information.) The whole environment is loosely coupled with one or more data warehouses managed by external storage systems. These systems can be OLAP servers (relational or multidimensional) and/or database management systems. The only hypothesis we make is that the basic multidimensional aspects of data analysis can be identified in the data warehouse representation provided by each system.

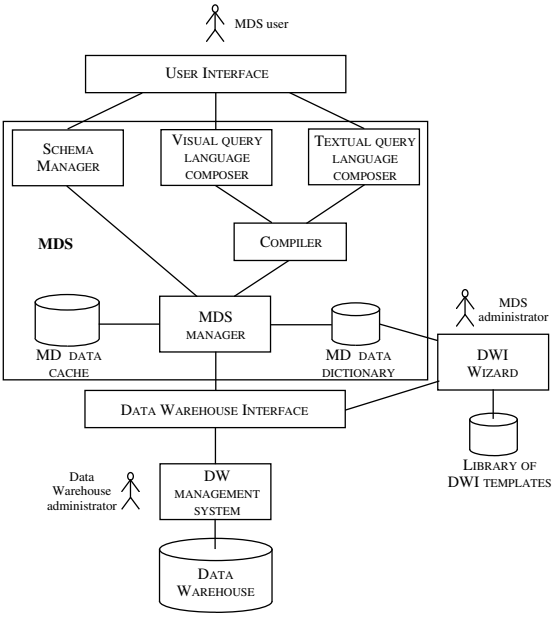


Fig. 2. The MDS system environment

The *Data Warehouse Interface* (DWI) performs two important functions: (i) it realizes the mapping between an $\mathcal{MD}$ database scheme and the actual organization of a data warehouse, and (ii) it converts $\mathcal{MD}$ queries into operations understandable by the underlying data warehouse server.

The *DWI Wizard* supports the MDS Manager with the construction of a DWI. This tool is separated from MDS but integrated with it. The DWI Wizard manages a library of DWI *templates* that refer to a specific database server but to a generic database. A DWI template can be customized, using the DWI Wizard, by specifying the actual scheme of a data warehouse and its correspondence with an $\mathcal{MD}$ database.

The MDS *Manager* is the core of the system and performs several tasks: (i) it imports the scheme of a data warehouse from a data storage system through the corresponding DWI and stores it into a local $\mathcal{MD}$ Data Dictionary; (ii) it receives $\mathcal{MD}$ queries (in an algebraic form) against a stored $\mathcal{MD}$ scheme, and sends them to the DWIs for execution; (iii) it receives the result of a query from one or more DWIs, possibly combines them, and then gives the final result to the Scheme Manager for its visualization; (iv) it receives the request for a creation of a view and stores its definition in the $\mathcal{MD}$ Data Dictionary. Query evaluation is supported by the management of a local cache of $\mathcal{MD}$ data.

The *Scheme Manager* requests the loading and translation into the $\mathcal{MD}$ model of an external data warehouse scheme, transfers the scheme of an available $\mathcal{MD}$ database to the User Interface for visualization, and allows the definition of views on it.

Relations

$R_t(\underline{Date},\ Month,\ Quarter)$ $\quad$ $R'_t(\underline{Quarter},\ Year)$

$R_p(\underline{Item},\ Category,\ Brand)$ $\quad$ $R_l(\underline{Store},\ Address,\ City,\ Area)$

$R_S(\underline{Item,\ Date,\ Store},\ Nsales,\ Revenue)$ $\quad$ $R_C(\underline{Item,\ Month},\ Cost)$

Functional dependencies (besides keys)

$R_t.Month \overset{FD}{\rightarrow} R_t.Quarter,$

$R_l.City \overset{FD}{\rightarrow} R_l.Area$

Foreign keys

$R_S.Store \overset{FK}{\rightarrow} R_l,\ R_S.Item \overset{FK}{\rightarrow} R_p,\ R_S.Date \overset{FK}{\rightarrow} R_t$

$R_C.Item \overset{FK}{\rightarrow} R_p,\ R_C.Month \overset{FK}{\rightarrow} R_t.Month$

$R_t.Quarter \overset{FK}{\rightarrow} R'_t$

Fig. 3. Example of the `RelRetail` data warehouse scheme

The *Visual query composer* supports the user in the specification of $\mathcal{MD}$ visual queries and verifies their correctness. The *Textual query parser* accepts queries written in the $\mathcal{MD}$ textual language. Both components transform $\mathcal{MD}$ queries into an internal format. The query *Compiler* translates visual and textual $\mathcal{MD}$ queries (specified according to the internal representation) into $\mathcal{MD}$ algebra expressions. The compiler also performs some optimization, by mainly pushing-down selections and projections. The *User Interface* allows the interaction with the system through both textual and graphical tools. The work of the user is supported by a number of menus and forms.

4 Data Warehouse Interfaces

Let M be the model adopted by a data warehouse server DS. Basically, a DWI for DS has three components:

1. a *Mapping* Θ, which associates concepts of a data warehouse scheme S of M with concepts of an $\mathcal{MD}$ scheme $\widehat{S}$;
2. a *Query Translation Algorithm* (**QTA**), which converts a query over an $\mathcal{MD}$ scheme into a query over a scheme of M that DS can understand; and
3. a *Data Translation Algorithm* (**DTA**), which transforms the results of queries and views over the scheme S into $\mathcal{MD}$ instances.

Both the algorithms used in a DWI make use of the mapping Θ and depend only on the features of DS. Therefore, once DS, and thus M, has been fixed, the only component that needs to be changed in the evolution of the data warehouse is the mapping Θ. The idea of DWI template is actually based on this consideration: we can fix the algorithms and obtain a new DWI for a scheme of M by just modifying Θ, as long as the organization of the data warehouse changes.

In order to give more insights on the structure of a DWI and on the way in which it works, we illustrate next an example that refers to a specific but important case. Specifically, we describe a possible DWI for a relational data warehouse organized according to (variants of) the star scheme. This is indeed a quite standard representation of a data warehouse in a ROLAP server.

Dimensions

$$\textbf{Product}=\{\ \langle \text{item}\rangle \overset{\Theta}{\leftrightarrow} \langle R_p.Item\rangle,\ \langle \text{category}\rangle \overset{\Theta}{\leftrightarrow} \langle R_p.Category\rangle,\ \langle \text{brand}\rangle \overset{\Theta}{\leftrightarrow} \langle R_p.Brand\rangle\ \}$$

$$\textbf{Location}=\{\ \langle \text{store}\rangle \overset{\Theta}{\leftrightarrow} \langle R_l.Store\rangle,\ \langle \text{city}\rangle \overset{\Theta}{\leftrightarrow} \langle R_l.City\rangle,\ \langle \text{area}\rangle \overset{\Theta}{\leftrightarrow} \langle R_l.Area\rangle\ \}$$

$$\textbf{Time}=\{\ \langle \text{day}\rangle \overset{\Theta}{\leftrightarrow} \langle R_t.Date\rangle,\ \langle \text{month}\rangle \overset{\Theta}{\leftrightarrow} \langle R_t.Month\rangle,\ \langle \text{quarter}\rangle \overset{\Theta}{\leftrightarrow} \langle R_t.Quarter\rangle,\ \langle \text{year}\rangle \overset{\Theta}{\leftrightarrow} \langle R'_t.Year\rangle\ \}$$

Roll-up functions

$$\langle \text{item} \overset{\rho}{\rightarrow} \text{category}\rangle \overset{\Theta}{\leftrightarrow} \langle R_p.Item \overset{FD}{\rightarrow} R_p.Category\rangle,\ \ \langle \text{item} \overset{\rho}{\rightarrow} \text{brand}\rangle \overset{\Theta}{\leftrightarrow} \langle R_p.Item \overset{FD}{\rightarrow} R_p.Brand\rangle$$

$$\langle \text{store} \overset{\rho}{\rightarrow} \text{city}\rangle \overset{\Theta}{\leftrightarrow} \langle R_l.Store \overset{FD}{\rightarrow} R_l.City\rangle,\ \ \langle \text{city} \overset{\rho}{\rightarrow} \text{area}\rangle \overset{\Theta}{\leftrightarrow} \langle R_l.City \overset{FD}{\rightarrow} R_l.Area\rangle$$

$$\langle \text{day} \overset{\rho}{\rightarrow} \text{month}\rangle \overset{\Theta}{\leftrightarrow} \langle R_t.Day \overset{FD}{\rightarrow} R_t.Month\rangle,\ \ \langle \text{month} \overset{\rho}{\rightarrow} \text{quarter}\rangle \overset{\Theta}{\leftrightarrow} \langle R_t.Month \overset{FD}{\rightarrow} R_t.Quarter\rangle,$$

$$\langle \text{quarter} \overset{\rho}{\rightarrow} \text{year}\rangle \overset{\Theta}{\leftrightarrow} \langle R_t.Quarter \overset{FK}{\rightarrow} R'_t, R'_t.Quarter \overset{FD}{\rightarrow} R'_t.Year\rangle$$

F-tables

$$\textsc{Sales} \overset{\Theta}{\leftrightarrow} R_S \qquad \textsc{CostOfItem} \overset{\Theta}{\leftrightarrow} R_C$$

$$\textsc{Sales}[\langle Period : \text{day}\rangle \overset{\Theta}{\leftrightarrow} \langle R_S.Date \overset{FK}{\rightarrow} R_t\rangle, \langle Product : \text{item}\rangle \overset{\Theta}{\leftrightarrow} \langle R_S.Item \overset{FK}{\rightarrow} R_p\rangle,$$
$$\langle Location : \text{store}\rangle \overset{\Theta}{\leftrightarrow} \langle R_S.Store \overset{FK}{\rightarrow} R_l\rangle] \rightarrow$$
$$[\langle NSales : \text{numeric}\rangle \overset{\Theta}{\leftrightarrow} \langle R_S.Nsales\rangle, \langle Income : \text{numeric}\rangle \overset{\Theta}{\leftrightarrow} \langle R_S.Revenue\rangle]$$

$$\textsc{CostOfItem}[\langle Product : \text{item}\rangle \overset{\Theta}{\leftrightarrow} \langle R_C.Item \overset{FK}{\rightarrow} R_p\rangle, \langle Month : \text{month}\rangle \overset{\Theta}{\leftrightarrow} \langle R_C.Month \overset{FK}{\rightarrow} R_t.Month\rangle] \rightarrow$$
$$[\langle Cost : \text{numeric}\rangle \overset{\Theta}{\leftrightarrow} \langle R_C.Cost\rangle]$$

Fig. 4. Example of mapping embedded in a relational DWI

A Relational DWI. Let us consider the relational star scheme `RelRetail` for a data warehouse, shown in Fig. 3. Note that this scheme is a possible relational implementation of the $\mathcal{MD}$ scheme `Retail` described in Fig. 1. In the scheme, R_S stores the daily sales and R_C the monthly costs of the items sold: they play the role of fact tables. The other tables correspond to dimensional relations: R_p stores item information, R_l location information and both R_t and R'_t store temporal information. Note that the scheme is not normalized. Also, we have indicated the integrity constraints defined on the scheme: they suggest the way to derive $\mathcal{MD}$ objects from the underlying relational database.

The mapping Θ for the `RelRetail` scheme is sketched in Fig. 4. This mapping associates: (i) an $\mathcal{MD}$ level with an attribute of a dimensional table, (ii) a roll-up function with a one or more relational integrity constraints, (iii) an $\mathcal{MD}$ f-table with a fact table, (iv) coordinates of an $\mathcal{MD}$ f-table with attributes and foreign keys from a fact table to a dimensional table, and (v) measures of an $\mathcal{MD}$ f-table with attributes of a fact table.

The mapping can be indeed used in both directions: on one hand it provides a basis for the translation of an $\mathcal{MD}$ query into a relational query, on the other hand it indicates how a relational scheme can be expressed in terms of the $\mathcal{MD}$ model. Actually, a Data Translation Algorithm can be easily derived from the mapping Θ and for this reason it will not be shown.

Let us consider now the **QTA** algorithm that translates queries over an $\mathcal{MD}$ scheme into SQL queries over a relational data warehouse server. In the algorithm, reported in Fig. 5, we make use of a four-tuple, which we call SQL

SQL Query Translation Algorithm: **SQL-QTA**
Input: An expression E of the $\mathcal{MD}$ algebra
Output: An SQL query descriptor δ

```
begin
    δ_s := ∅;   δ_f := ∅;   δ_w := ∅;   δ_g := ∅;
    case E of
      E is an f-table F :
        δ_s := {the coordinates and measures of F, according to the mapping Θ};
        δ_f := {the relation representing F, according to the mapping Θ};
      E is a selection σ_C(E') :
        δ' := SQL-QTA(E');   δ_s := δ'_s;
        if δ' is not aggregative then
            δ_f := δ'_f;
            δ_w := δ'_w ∪ {the conditions corresponding to C, according to the mapping Θ}
        else
            δ_f := {δ'};
            δ_w := {the conditions corresponding to C, according to the mapping Θ}
        endif
      E is a projection π_X(E') :
        δ' := SQL-QTA(E');
        δ_s := {the coordinates and measures of X, according to the mapping Θ};
        δ_f := δ'_f;   δ_w := δ'_w;   δ_g := δ'_g
      E is an equi-join E' ⋈_X E'' :
        δ' := SQL-QTA(E');   δ'' := SQL-QTA(E'');
        δ_s := δ'_s ∪ δ''_s;
        if both δ' and δ'' are not aggregative then
            δ_f := δ'_f ∪ δ''_f;
            δ_w := δ'_w ∪ δ''_w ∪ {the conditions corresponding to X, according to the mapping Θ}
        else
            δ_f := {δ'} ∪ {δ''};
            δ_w := {the conditions corresponding to X, according to the mapping Θ}
        endif
      E is a scalar function application φ^{M=f(...)}(E') :
        δ' := SQL-QTA(E');
        δ_s := δ'_s ∪ {f(...) AS M, according to the mapping Θ};
        δ_f := δ'_f;   δ_w := δ'_w;   δ_g := δ'_g
      E is an aggregate function application ψ_X^{N=g(...)}(E') :
        δ' := SQL-QTA(E');
        δ_s := {g(...) AS N, according to the mapping Θ} ∪
            {the attributes and measures of X, according to the mapping Θ};
        if δ' is not aggregative then
            δ_f := δ'_f;   δ_w := δ'_w;
            δ_g := {the attributes and measures of X, according to the mapping Θ}
        else
            δ_f := {δ'};
            δ_g := {the attributes and measures of X, according to the mapping Θ}
        endif
      E is a roll-up expression Q_l^{l'}(E') :
        δ' := SQL-QTA(E');
        δ_s := δ'_s ∪ {the attributes corresponding to l and l', according to the mapping Θ};
        if δ' is not aggregative then
            δ_f := δ'_f ∪ {the relations corresponding to Q_l^{l'}, according to the mapping Θ};
            δ_w := δ'_w ∪ {the conditions corresponding to Q_l^{l'}, according to the mapping Θ}
        else
            δ_f := {δ'} ∪ {the relations corresponding to Q_l^{l'}, according to the mapping Θ};
            δ_w := {the conditions corresponding to Q_l^{l'}, according to the mapping Θ}
        endif
    endcase;
    return (δ_s, δ_f, δ_w, δ_g);
end
```

Fig. 5. The query translation algorithm embedded in a relational DWI

query descriptor, to represent the `SELECT`, `FROM`, `WHERE`, and `GROUP BY` clauses of an SQL query, respectively. According to SQL syntax, in a query descriptor $\delta = (\delta_s, \delta_f, \delta_w, \delta_g)$, δ_s denotes a set of (expressions on) attributes possibly involving aliases, δ_f denotes a set of relation names and query descriptors that indicate nested SQL queries, δ_w denotes a set of conditions of the form $A_1 = A_2$ (where A_1 and A_2 are attributes), and δ_g is a subset of the attributes occurring in S. A query descriptor is *aggregative* if the set δ_s contains an aggregate function.

According to the structure of MDS, the algorithm takes as input a query expressed in $\mathcal{MD}$ algebra and returns a query descriptor as output, which is then transformed into an SQL query. The algorithm operates recursively on the structure of the expression in input and transforms elements of the $\mathcal{MD}$ algebra into elements of relational algebra, according to the mapping Θ.

The basic step corresponds to the case in which an f-table F is encountered (first option of the case statement). In this case δ_s takes the relational attributes associated with the coordinates and the measures of F and δ_f takes the fact table associated with F (see Fig. 4).

Then, at each step, the query under construction is either extended or nested (in the `FROM` clause); this depends on whether an aggregation has been encountered. For instance, when a selection $\sigma_C(E')$ is encountered (second option of the case statement) and E' does not involve an aggregation, δ_w is extended with the condition obtained from C by substituting coordinates and measures with the relational attributes associated with them, according to Θ. Conversely, if E' is aggregative, δ_f is replaced by the query descriptor obtained by applying the translation algorithm to E'. This corresponds to a nesting in the `FROM` clause. The other cases are similar.

A special and interesting case is when a roll-up expression $\varrho_l^{l'}(E')$ is encountered (last option of the case statement). If E' is not aggregative, δ_f is extended with the tables involved in the integrity constraints that Θ associates with the roll-up function R-UP$_l^{l'}$, whereas δ_w is extended with the join conditions corresponding to the foreign keys involved (see Fig. 4). If E' is instead aggregative, the construction proceeds as above.

The formal nature of the $\mathcal{MD}$ model and language allows us to investigate formal issues about the expressive power of DWI's with respect to query capabilities of the underlying OLAP servers.

References

1. L. Cabibbo and R. Torlone. A logical approach to multidimensional databases. In *EDBT'98, Springer-Verlag*, pag. 183–197, 1998.
2. L. Cabibbo and R. Torlone. From a procedural to a visual query language for OLAP. In *SSDBM'98*, IEEE Computer Society Press, pag. 74–83, 1998.
3. S. Chaudhuri and U. Dayal. An overview of data warehousing and OLAP technology. *ACM SIGMOD Record*, 26(1):65–74, March 1997.
4. W.H. Inmon. *Building the data warehouse*. John Wiley & Sons, 1996.
5. R. Kimball. *The data warehouse toolkit*. John Wiley & Sons, 1996.
6. N. Pendse and R. Creeth. The OLAP report. *http://www.olapreport.com*.
7. *Microsoft OLE DB programmer's reference and data access SKD*. Microsoft Corporation, 1998.

Applying Vertical Fragmentation Techniques in Logical Design of Multidimensional Databases[1]

Matteo Golfarelli, Dario Maio, and Stefano Rizzi

DEIS - University of Bologna,
Viale Risorgimento 2, 40136 Bologna, Italy
{mgolfarelli, dmaio, srizzi}@deis.unibo.it

Abstract. In the context of multidimensional databases implemented on relational DBMSs through star schemes, the most effective technique to enhance performances consists of materializing redundant aggregates called views. In this paper we investigate the problem of vertical fragmentation of views aimed at minimizing the workload response time. Each view includes several measures which not necessarily are always requested together; thus, the system performance may be increased by partitioning the views into smaller tables. On the other hand, drill-across queries involve measures taken from two or more views; in this case the access costs may be decreased by unifying these views into larger tables. After formalizing the fragmentation problem as a 0-1 integer linear programming problem, we define a cost function and outline a branch-and-bound algorithm to minimize it. Finally, we demonstrate the usefulness of our approach by presenting a set of experimental results based on the TPC-D benchmark.

1 Introduction

Recently, multidimensional databases have gathered wide research and market interest as the core of decision support applications such as data warehouses [1][11]. A *multidimensional database* (MD) can be seen as a collection of multidimensional "cubes" centered on facts of interest (for instance, the sales in a chain store); within a cube, each cell contains information useful for the decision process, i.e., a set of numerical *measures*, while each axis represents a possible *dimension* for analysis.

An MD implemented on a relational DBMS is usually organized according to the so-called *star scheme* [13], in which each cube is represented by one *fact table* storing the measures and one denormalized *dimension table* for each dimension of analysis. The primary key of each dimension table (usually a *surrogate key*, i.e., internally generated) is imported into the fact table; the primary key of the fact table is defined by the set of these foreign keys. Each dimension table contains a set of *attributes* defining a hierarchy of aggregation levels for the corresponding dimension.

The basic mechanism to extract useful information from elemental data in MDs is aggregation. In order to improve the system performance for a given workload, an MD typically stores, besides the elemental values of measures, also values summarized according to some *aggregation patterns*, i.e., sets of attributes taken from dimension

[1] This research was partially supported by MURST - Interdata Project.

Y. Kambayashi, M. Mohania, and A M. Tjoa (Eds.): DaWaK 2000, LNCS 1874, pp. 11–23, 2000.
© Springer-Verlag Berlin Heidelberg 2000

tables to define the coarseness of aggregation. Even data summarized according to each pattern are organized into a star scheme, whose fact table is called a *view* and imports the attributes included in the pattern; measure values are obtained by applying an aggregation operator to the data in another fact table with a finer pattern. In the following we will use the term *view* to denote either the fact tables containing elemental values (*primary views*) or those containing aggregated values (*secondary views*).

Since pre-computing all the possible secondary views is unfeasible, several techniques have been proposed to select the subset of views to materialize in order to optimize the response to the workload (e.g, [3][12][19]). In this paper we investigate how the response can be further enhanced by fragmenting views vertically. By vertical fragmentation we mean the creation of fragments of views, each including measures taken from one or more views with the same pattern as well as the key associated to that pattern. Fragmentation may achieve two goals together: partitioning the measures of a view into two or more tables, and unifying two or more views into a single table. While partitioning may be useful whenever only a subset of the attributes is typically required by each query, unification may be convenient when the workload is significantly affected by drill-across queries, i.e., queries formulated by joining two or more views deriving from different cubes.

It is remarkable that partitioning entails no significant storage overhead. In fact, on the one hand, surrogate keys require a few bytes to be stored. On the other, though on primary views the number of dimensions may exceed the number of measures, this is less likely on secondary views for two reasons. Firstly, it may be necessary to include in them also derived measures and support measures for non distributive aggregation operators [9][10]. Secondly, within a coarse aggregation pattern, one or more dimensions may be completely aggregated (in this case, the corresponding foreign key is dropped from the key of the view).

As compared to operational databases, in MDs the benefits of fragmentation are further enhanced by the multiple query execution plans due to the presence of redundant secondary views. These benefits are particularly relevant if the MD is implemented on a parallel architecture; if disk arrays are adopted and fragmentation is coupled with an allocation algorithm, the queries requiring multiple fragments allocated on different disks can be effectively parallelized [15][16].

The problem of determining the optimal partitioning given a workload has been widely investigated within the context of centralized as well as distributed database systems, considering non-redundant allocation of fragments (for instance, see [6][14][16]); unfortunately, the results reported in the literature cannot be applied here since the redundancy introduced by materializing views binds the partitioning problem to that of deciding on which view(s) each query should be executed. To the best of our knowledge, the problem of vertical fragmentation in MDs has been dealt with only in [15], where no algorithm for determining the optimal fragmentation is proposed. In [7], views are partitioned vertically in order to build *dataindices* to enhance performance in parallel implementations of MDs.

In Section 2 we outline the necessary background for the paper. In Section 3 the vertical fragmentation problem is formalized, a cost function is proposed and a branch-and-bound approach is proposed. Section 4 presents some results based on the TPC-D benchmark.

2 Background

2.1 Cubes and Patterns

In this paper we will often need to refer to multidimensional objects from a conceptual point of view, apart from their implementation on the logical level. For this reason we introduce *cubes*.

A multidimensional cube f is characterized by a set of dimensions $Patt(f)$, a set of measures $Meas(f)$ and a set of attributes $Attr(f)$ $Patt(f)$. The attributes in $Attr(f)$ are related into a directed acyclic graph by a set of functional dependencies a_i a_j. From now on, by writing a_i a_j we will denote both the case in which a_i directly determines a_j and that in which a_i transitively determines a_j. It is required that

; .

The cube we will use as a working example, *LineItem*, is inspired by a star scheme in the TPC-D [18] and describes the composition of the orders issued to a company; it is characterized by:

$Patt(LineItem)$ = {Part, Supplier, Order, ShipDate},
$Meas(LineItem)$ = {Price, Qty, ExtPrice, Discount, DiscPrice, SumCharge, Tax}

and by the attributes and functional dependencies shown in Fig. 1, where circles represent attributes (in gray the dimensions).

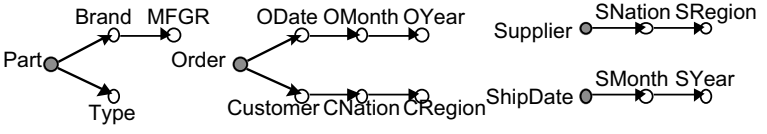


Fig. 1. Functional dependencies in the *LineItem* cube

On relational DBMSs, cubes are usually implemented adopting the star scheme. The star scheme for *LineItem* is:

PART (PartId, Part, Brand, MFGR, Type)
SUPPLIER (SupplierId, Supplier, SNation, SRegion)
ORDER (OrderId, Order, ODate, OMonth, OYear, Customer, CNation, CRegion)
SHIPDATE (ShipDateId, ShipDate, SMonth, SYear)
LINEITEM (PartId, SupplierId, OrderId, ShipDateId, Price, Qty, ExtPrice, Discount, DiscPrice, SumCharge, Tax)

where LINEITEM is the primary view; the other tables are dimension tables. For the sake of simplicity, we will not consider the possibility of normalizing dimension tables to obtain snowflake schemes.

Definition 1. Given a cube f, an *aggregation pattern* (or simply *pattern*) on f is a set P $Attr(f)$ such that no functional dependency exists between each pair of attributes in P: .

With reference to the *LineItem* cube, examples of patterns are *Patt*(*f*), {Part, OMonth, SNation}, {Brand, Type}, {}.

Definition 2. Let P_i and P_j be two patterns; we say that P_i is *coarser* than P_j (P_i P_j) if .

For instance, {Brand, CRegion} {Brand, Customer, Supplier}.

2.2 The Workload

In principle, the workload for a MD is dynamic and unpredictable. A possible approach to cope with this fact, adopted in some commercial tools, consists of monitoring the actual workload while the MD is operating. Otherwise, the designer may try to determine a core workload a priori: in fact, on the one hand, the user typically knows in advance which kind of data analysis (s)he will carry out more often for decisional or statistical purposes; on the other, a substantial amount of queries are aimed at extracting summary data to fill standard reports.

As to update queries, we believe they should not be included in the workload. In fact, MDs are typically updated only periodically, in an off-line fashion, and during this process the database is unavailable for querying. Thus, the update process does not directly affect the MD performance, and it is sufficient to ensure that it is properly bounded in time.

Definition 3. The workload is a set of pairs (q_i, $_i$), where q_i denotes a query and $_i$ its expected frequency.

Within the scope of this paper, a query q can be characterized by (1) its pattern, *Patt*(q); (2) the set of measures it requires, *Meas*(q); (3) the selectivity, *sel*(q), defined as the ratio between the number of tuples returned by q and the cardinality of the view at *Patt*(q). For instance, on *LineItem*, the query asking for the total quantity of each medium polished part ordered from each American supplier is characterized by *Patt*(q) = {Supplier, Part}, *Meas*(q) = {Qty} and *sel*(q) = 0.01 (assuming that 5 supplier regions and 20 part types are present, that attribute values are uniformly distributed and that selection predicates are independent, it is *sel*(q) = — —).

Part of the queries the user formulates may require comparing measures taken from distinct, though related, cubes; in the OLAP terminology, these are called drill-across queries.

Definition 4. Let $f_1,...f_m$ be m cubes such that — — ;
a *drill-across query* on $f_1,...f_m$ is a query q characterized by *Patt*(q) = — and , *Meas*(q) *Meas*(f_i) for i=1,...m. We call the *projection* of q on cube f_i the query q_i characterized by *Patt*(q_i) = — and *Meas*(q_i) = *Meas*(q) *Meas*(f_i).

Consider for instance the cube *Shipment* characterized by:

Patt(*Shipment*) = {Part, ShipTo, ShipFrom, ShipMode, ShipDate},
Meas(*Shipment*) = {QtyShipped, ShippingCost}

and by the attributes and functional dependencies shown in Fig. 2. Since pattern P = {Part, Customer, ShipDate} and all the other patterns coarser than P are common to *LineItem* and *Shipment*, a possible drill-across query is the one asking for the total cost paid by the customers of each region to receive each part, characterized by $Patt(q)$ = {CRegion, Part}, $Meas(q)$ = {DiscountPrice, ShippingCost} and $sel(q) = 1$.

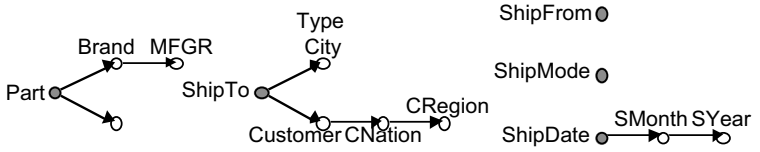


Fig. 2. Functional dependencies in the *Shipment* cube

2.3 Views

Given a cube f, each pattern on f determines a candidate view for materialization. Several algorithms have been proposed to determine the optimal set of views to be materialized, often by significantly reducing the search space [3][12]. Discussing these algorithms is outside the scope of this paper; we will assume that one of them is applied to determine, for each cube, an optimal set of views. To the best of our knowledge, no workload-based materialization algorithms in the literature takes drill-across queries into account; on the other hand, since these queries play a relevant role within our workload, it is necessary to involve them in the optimization process. Thus, when applying the materialization algorithm, every drill-across query is substituted by its projections on the cubes involved.

Let V be the global set of the (primary and secondary) views to be materialized for the MD considered, as determined by the materialization algorithm. Given view $v \in V$, we will denote with $Patt(v)$ and $Meas(v)$, respectively, the pattern on which v is defined (determined by its primary key) and the set of measures it contains. The primary view for cube f is characterized by $Patt(v) = Patt(f)$ and $Meas(v) = Meas(f)$; the secondary views for f by $Patt(v) \supset Patt(f)$ and $Meas(v) = Meas(f)$.

Given query q and view v, q can be answered on v if $Patt(q) \subseteq Patt(v)$ and $Meas(q) \subseteq Meas(v)$. In particular:

If q involves only measures from a single cube f, it can always be answered on the primary view for f or, more conveniently, on any secondary view v for f provided that $Patt(q) \subseteq Patt(v)$.

If q is a drill-across query on cubes $f_1,...f_m$, by definition it is $Patt(q) \subseteq Patt(f_i)$ for each i; thus, q can be solved by first solving all the projections of q on the m cubes, then performing a join between the results (the join attributes are those in $Patt(q)$).

3 Vertical Fragmentation of Views

Each view includes measures which describe the same cube but, within the workload, may be often requested separately. Thus, the system overall performance may be increased by *partitioning* the views determined from the materialization algorithm into smaller tables, each including only the measures which typically appear together within the queries. On the other hand, drill-across queries can be solved by joining views defined on different cubes. The access costs for these queries may be decreased by *unifying* two or more views on the same pattern into larger tables where all the measures required are stored together.

With the term *fragmentation* we denote both partitioning and unification of (primary or secondary) views. The approach we propose is aimed at determining an optimal fragmentation of the views in the set V.

It is remarkable that the effectiveness of fragmentation for MDs may be higher than for operational non-redundant databases; in fact, while in the latter case it is known a priori on which table(s) each query will be executed, in MDs the presence of redundant views makes multiple solutions possible. In the following we consider an example on *LineItem*. Let $V = \{v_1, v_2\}$, where

$Meas(v_1) = Meas(LineItem)$; $Patt(v_1) = \{\text{SNation, Brand}\}$
$Meas(v_2) = Meas(LineItem)$; $Patt(v_2) = \{\text{SNation, Part, ODate}\}$

Let the workload include two queries q_1 and q_2 defined as follows:

$Meas(q_1) = \{\text{Price, Qty, Discount, ExtPrice, DiscPrice}\}$; $Patt(q_1) = \{\text{SNation, Brand}\}$
$Meas(q_2) = \{\text{Tax, DiscPrice, SumCharge}\}$; $Patt(q_2) = \{\text{SNation, Brand}\}$

It is convenient to execute both q_1 and q_2 on v_1 since its cardinality is lower than that of v_2 ($Patt(v_1) > Patt(v_2)$). Now, consider a fragmentation including four fragments:

$Meas(v'_1) = \{\text{Price, Qty, Discount, ExtPrice, DiscPrice}\}$; $Patt(v'_1) = Patt(v_1)$
$Meas(v''_1) = \{\text{Tax, SumCharge}\}$; $Patt(v''_1) = Patt(v_1)$
$Meas(v'_2) = \{\text{DiscPrice}\}$; $Patt(v'_2) = Patt(v_2)$
$Meas(v''_2) = \{\text{Price, Qty, Discount, ExtPrice, Tax, SumCharge}\}$; $Patt(v''_2) = Patt(v_2)$

This solution is optimal for q_1, which will be executed on v'_1. As to q_2, while Tax and SumCharge are retrieved from v''_1, it might be more convenient to retrieve DiscPrice from v'_2 rather than from v'_1, depending on the trade-off between reading less measures and accessing less tuples. In general, another factor to be considered in the trade-off is the number of attributes forming the fact table key: for coarser patterns, the length of the key is shorter and the size of the tuples read is smaller. The possibility of answering a query by jointly accessing fragments of different patterns, impacts on the optimization of the query execution tree by enabling additional push-downs and pull-ups of group-by operators.

3.1 Problem Statement

In principle, the fragmentation algorithm should be applied to the whole set of views, V. On the other hand, it may be convenient to unify two measures belonging to two different cubes f' and f'' only if at least two views with the same pattern have been materialized on f' and f'' and the workload includes at least one drill-across query on f'

and *f"* which can be answered on these two views; in this case, we say that *f'* and *f"* are *strictly related*. Two cubes are *related* if (1) they are strictly related or (2) a third cube related to both exists. The transitive notion of relatedness induces a partitioning onto the set of cubes belonging to the MD, which in turn partitions the set of queries and the set of views according to the cube(s) they are defined on; in order to decrease complexity, fragmentation is meant to be applied separately to each set of queries on the corresponding set of related cubes.

Let *FS* be a set of related cubes and *QS* be the set of queries on the cubes in *FS*. Let *VS* *V* be the set of views materialized on the cubes in *FS* and *PS* be the set of patterns characterizing the views in *VS*.

Definition 5. Given cube *f* *FS*, we partition[2] *Meas*(*f*) into the largest subsets of measures which appear all together in at least one query of *QS* and do not appear separately in any other query in *QS*:

.

We call each subset a *minterm* of *f*, and denote with *MS*(*f*) the set of all minterms of *f*.

For instance, on the *LineItem* cube, given $QS = \{q_1, q_2\}$ where $Meas(q_1)$ = {Price, Qty, ExtPrice, Discount} and $Meas(q_2)$ = {Price, Qty, DiscPrice, SumCharge}, it is *MS*(*LineItem*) = {{Price, Qty}, {ExtPrice, Discount}, {DiscPrice, SumCharge}}.

Definition 6. Given the set of related cubes *FS*, we define a *term* as a set of measures which either is a minterm of a cube in *FS* or is the union of the minterms, even from different cubes in *FS*, within a set such that . We denote with *TS* the set of terms for *FS*.

For example above, it is *TS* = *MS*(*LineItem*) {*Meas*(q_1), *Meas*(q_2)}.

Given *FS*, a solution to the fragmentation problem is encoded by a *fragmentation array*, i.e., a binary array $\boldsymbol{C}$ with three dimensions corresponding to, respectively, the queries q_i *QS*, the patterns P_j *PS* and the terms T_k *TS*. The set of *fragments* defined by $\boldsymbol{C}$ is

where fragment v_{jk} is characterized by $Meas(v_{jk}) = T_k$ and $Patt(v_{jk}) = P_j$.

A fragmentation array not only denotes a fragmentation of the views in *VS*; at the same time, it specifies on which fragment(s) each query is assumed to be executed. In fact, a 1 in cell C_{ijk} denotes that, when answering query q_i, the measures in $Meas(q_i)$ T_k will be obtained from v_{jk}.

The fragmentation encoded by $\boldsymbol{C}$ is *feasible* if the following constraints are satisfied:

[2] We assume that each measure appears in at least one query of the workload.

(1) For each query, every measure required must be obtained from exactly one fragment (non ambiguous query execution):

, .
; ;

(2) For each pattern, each measure must belong to at most one fragment (non redundant fragmentation):

, , ; .

(3) Each fragment in *VS'* must be a fragmentation of one or more views in *VS* (consistency with the views materialized):

, ; ; .

Though constraint (3) states that every fragment must derive from a view in *VS*, we are not guaranteed that every measure in every view is included in a fragment. In fact, fragments will be produced only for the minterms, among those deriving from each view, actually used to answer at least one query. Given a view, we will call *lost minterms* those not included in any term generating a fragment. The fragmentation of primary views must necessarily be lossless, thus, every lost primary minterm must be reconsidered *a posteriori*, either by creating a separate fragment or by unifying it with one of the fragments determined. On the other hand, as to lost minterms from secondary views, it is not obvious whether generating the fragments is convenient or not; in fact, the space saved could probably be more profitably employed to store indices or additional views.

In the following we consider a small example on $FS = \{LineItem, Shipment\}$. Let $QS = \{q_1, q_2, q_3, q_4, q_5\}$; q_1, q_2, q_3 are defined on *LineItem*, q_4 on *Shipment* and q_5 is a drill-across query:

$Meas(q_1)$ = {Price, Qty, Discount}; $Patt(q_1)$ = {ShipDate}
$Meas(q_2)$ = {ExtPrice, DiscPrice}; $Patt(q_2)$ = {Part, Customer}
$Meas(q_3)$ = {SumCharge, Tax}; $Patt(q_3)$ = {Part, CNation}
$Meas(q_4)$ = {QtyShipped, ShippingCost}; $Patt(q_4)$ = {CNation, MFGR, ShipDate}
$Meas(q_5)$ = {ExtPrice, DiscPrice, ShippingCost}; $Patt(q_5)$ = {Brand, CNation}

We assume that, besides the primary views v_1 and v_2, two secondary views v_3 and v_4 have been materialized on *LineItem*, one secondary view v_5 on *Shipment*:

$Patt(v_3)$ = {Part, Customer}; $Patt(v_4) = Patt(v_5)$ = {Part, CNation}

(for each view, the measures are those of the corresponding cube). Fig. 3 shows the fragmentation array which represents a feasible solution to this fragmentation problem, which features five fragments:

$Meas(v'_1)$ = {Price, Qty, Discount}; $Patt(v'_1) = Patt(LineItem)$
$Meas(v'_2)$ = {QtyShipped, ShippingCost}; $Patt(v'_2) = Patt(Shipment)$
$Meas(v'_3)$ = {ExtPrice, DiscPrice}; $Patt(v'_3) = Patt(v_3)$
$Meas(v'_4)$ = {SumCharge, Tax}; $Patt(v'_4) = Patt(v_4)$
$Meas(v'_5)$ = {ExtPrice, DiscPrice, ShippingCost}; $Patt(v'_5) = Patt(v_4) = Patt(v_5)$

the first four of which are obtained by partitioning, the last one by coupling partitioning and unification. The array also denotes that, for instance, query q_1 is executed on v'_1.

TS \ *PS*	*Patt(LineItem)*	*Patt(Shipment)*	*Patt*(v_3)	*Patt*(v_4) = *Patt*(v_5)	*Patt(LineItem)*	*Patt(Shipment)*	*Patt*(v_3)	*Patt*(v_4) = *Patt*(v_5)	*Patt(LineItem)*	*Patt(Shipment)*	*Patt*(v_3)	*Patt*(v_4) = *Patt*(v_5)	*Patt(LineItem)*	*Patt(Shipment)*	*Patt*(v_3)	*Patt*(v_4) = *Patt*(v_5)	*Patt(LineItem)*	*Patt(Shipment)*	*Patt*(v_3)	*Patt*(v_4) = *Patt*(v_5)
{Discount,Qty,Price}	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
{DiscPrice,ExtPrice}	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
{SumCharge,Tax}	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
{ShippingCost}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
{QtyShipped}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
{ShippingCost,QtyShipped}	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
{DiscPrice,ExtPrice,ShippingCost}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
	q_1				q_2				q_3				q_4				q_5			

Fig. 3. Fragmentation array representing a feasible solution

3.2 The Cost Function

Among all the feasible solutions to the fragmentation problem, we are interested in the one which minimizes the cost for executing the workload. We believe it is convenient to keep logical design separate from the physical level in order to both provide a more general solution and reduce complexity; thus, the cost function we propose intentionally abstracts from any assumptions on the access paths, being based on the number of disk pages in which the tuples of interest for a given query are stored. In particular, the cost of query q_i within fragmentation $\boldsymbol{C}$ is defined as:

———

where:

$ns(P_j)$ is the cardinality for the view on pattern P_j (estimated for instance as shown in [8]).

$sel(q_i)$ is the selectivity of q_i; thus, $sel(q_i) \cdot ns(P_j)$ is the number of tuples of the view on pattern P_j which must be accessed in order to answer q_i.

$_{jk}$ is the number of tuples per disk page for fragment v_{jk} characterized by P_j and T_k:

———— ———————— .

Thus, ——— is the number of pages in which v_{jk} is contained.

, ——— is the expected number of pages in which the tuples necessary for q_i are stored, estimated with the Cardenas formula [5].

Thus, $cost(q_i,\mathbf{C})$ expresses the total number of disk pages which must be accessed in order to solve q_i. Though the actual number of pages read when executing the query may be higher depending on the access path followed, we believe that this function represents a good trade-off between generality and accuracy.

It should be noted that, whenever two views are unified, the resulting fragment may be used to answer not only drill-across queries, but also queries on the single cubes; thus, it must contain the union of their tuples. Let *f'* and *f"* be two cubes, let *P* be a pattern common to *f'* and *f"*, and let *ns'*(*P*) and *ns"*(*P*) be the cardinalities of views *v'* on *f'* and *v"* on *f"*, respectively. The cardinality of the view unifying *v'* and *v"* can be estimated as:

———————

where *cs*(*P*) is the product of the domain cardinalities for the attributes in *P*.

3.3 A Branch-and-Bound Approach

The problem of vertical fragmentation (*VFP*) can be formulated as follows: *Find, for the binary decision array* $\mathbf{C}$*, the value which minimizes function*

subject to constraints (1), (2), (3) expressed in Section 3.1. VFP is a 0-1 integer linear programming problem like *set covering with additional constraints*, and is known to be NP-hard [17]. Thus, a branch-and-bound approach can be adopted to solve it optimally.

The ingredients of a branch-and-bound procedure for a discrete optimization problem such as *VFP* are [1]:

(i) A *branching rule* for breaking up the problem into subproblems. Let *VFP* be the problem of choosing, given a partial solution to *VFP* represented by an "incomplete" array[3] $\mathbf{C}(VFP)$, the remaining elements C_{ijk} to be set to 1 in the complete solution. We denote with *SUB*(*VFP*) the set of subproblems in which *VFP* is broken up; each is defined by choosing one element C_{ijk} to be set to 1 in the partial solution, which means adding to the current solution a

[3] Non feasible since one or more queries cannot be answered (constraint (1) is not satisfied).

fragment on pattern P_j to be used for retrieving some measures T_k to solve query q_i.

(ii) A *subproblem selection rule* for choosing the next (most promising) subproblem to be processed. The element C_{ijk} chosen is the one for which $ns(P_j)$ is minimum and $Meas(q_i)$ T_k has maximum cardinality.

(iii) A *relaxation* of VFP, i.e. an easier problem VFR whose solution bounds that of VFP. We relax VFP by removing constraint (2): in VFR, some measures may be replicated in two or more fragments defined on the same pattern.

(iv) A *lower bounding* procedure to calculate the cost of the relaxation. VFR consists of one set covering problem for each q_i, which can be solved by adopting one of the algorithms in the literature [4]. Since in solving VFR the number of eligible fragments is higher than that for VFP, the cost of VFR will be lower or equal to that of VFP.

4 Experimental Tests

In this paper we have proposed an approach to vertical fragmentation of views in multidimensional databases. The experimental results we present in this section confirm the utility of the approach in terms of reduction of the cost for executing the expected workload. The tests we have carried out are based on the well-known TPC-D benchmark [18], which features two cubes *LineItem* and *PartSupplier* with cardinalities 6.000.000 and 800.000, respectively; the total amount of data is about 1 Gbyte.

We have tested our approach on the Informix DBMS with a workload based on the 17 TPC-D queries (all with the same frequency). The views to be fragmented have been selected by means of the heuristic approach to materialization proposed in [3], by considering a global space constraint of 2 GB (1 GB for primary views + 1 GB for secondary views); as a result, 11 secondary views were created besides the 2 primary views. The fragmentation algorithm determined 14 fragments (3 from the primary views) and 9 lost minterms. Indices on all the attributes belonging to keys in both fact and dimension tables were created.

Fig. 4 shows, for each query, the ratio between the number of disk pages read without and with fragmentation; above each column, the number of disk pages read without fragmentation. Overall, fragmentation decreases the workload cost from 265904 to 59986 pages (more than 4 times).

Fig. 5 shows how fragmentation affects the total storage space; above each column, the storage space without fragmentation. Overall, the unfragmented views require 368840 pages; while materializing only the fragments (no lost minterms) decreases the space required to 306042 pages (-17.0%), materializing also lost minterms increases the space required to 442097 pages (+19.8%).

It should be noted that the next view to be materialized beyond the 2 GB constraint would take 126460 disk pages, and decrease the workload cost by 1%; fragmentation is more convenient since it takes only 73257 extra pages and decreases the cost by 77%. In fact, while materializing one more view typically benefits few queries, several queries may take advantage from using the same disk space for fragmentation. Furthermore, while fragmentation does not require extra space for dimension tables, each new view may require adding new tuples in dimension tables to be referenced by the aggregated tuples in the view.

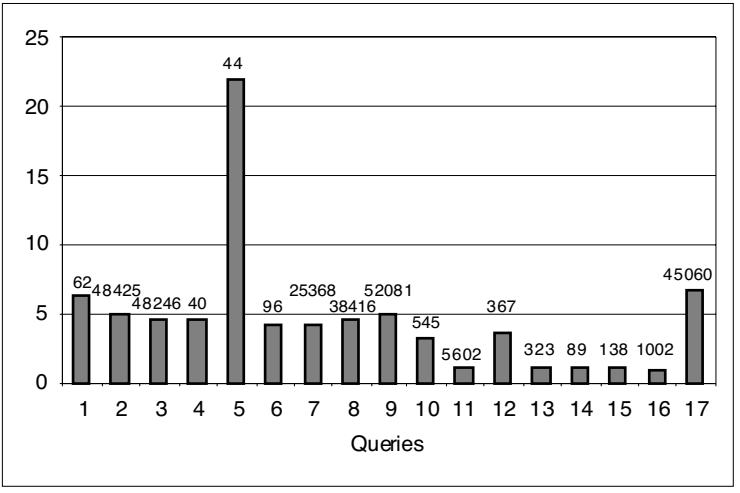


Fig. 4. Ratio between the number of disk pages read without and with fragmentation

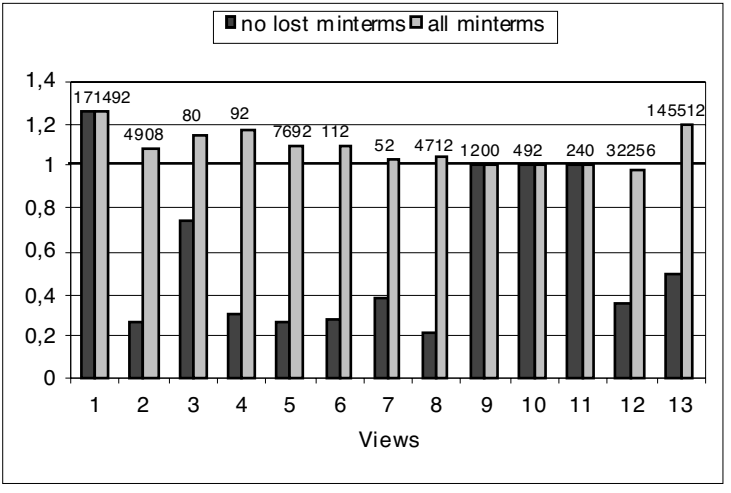


Fig. 5. Ratio between the storage space with and without fragmentation

In order to evaluate the algorithm complexity, we have defined four more workloads, each progressively extending the TPC-D; the results are shown in Table I. The computing time does not depend strictly on the workload size, of course it is also determined by the relationships between the queries.

Table I. Results of the complexity tests

n. queries in the workload	*n. subproblems generated*	*computing time*
17	2775	about 1 min
25	4439	about 2 mins
30	348925	about 30 mins
35	51099	about 12 mins
40	403420	about 75 mins

References

1. Balas, E., Toth, P.: Branch and bound methods. In: Lawler, E. et al. (eds.): The traveling salesman problem. John Wiley & Sons (1985) 361-397
2. Agrawal, R., Gupta, A., Sarawagi, S.: Modeling multidimensional databases. IBM Research Report (1995)
3. Baralis, E., Paraboschi, S., Teniente, E.: Materialized view selection in multidimensional database. Proc. 23rd Int. Conf. on Very Large Data Bases, Athens, Greece (1997) 156-165
4. Beasley, J.E.: An Algorithm for Set Covering Problems. European Journal of Operational Research 31 (1987) 85-93
5. Cardenas, A.F.: Analysis and performance of inverted database structures. Comm. ACM 18(5) (1975) 253-263
6. Chu, W.W., Ieong, I.T.: A transaction-based approach to vertical partitioning for relational database system. IEEE Trans. on Software Eng. 19(8) (1993) 804-812
7. Datta, A., Moon, B., Thomas, H.: A case for parallelism in data warehousing and OLAP. Proc. IEEE First Int. Workshop on Data Warehouse Design and OLAP Technology (1998)
8. Golfarelli, M., Rizzi, S.: Designing the data warehouse: key steps and crucial issues. Journal of Computer Science and Information Management 2(3) (1999)
9. Golfarelli, M., Rizzi, S.: View Materialization for Nested GPSJ Queries. To appear on Proc. DMDW'2000, Stockholm (2000)
10. Gray, J., Bosworth, A., Lyman, A., Pirahesh, H.: Data-Cube: a relational aggregation operator generalizing group-by, cross-tab and sub-totals. Technical Report MSR-TR-95-22, Microsoft Research (1995)
11. Gyssens, M., Lakshmanan, L.V.S.: A foundation for multi-dimensional databases. Proc. 23rd Very Large Database Conf., Athens, Greece (1997) 106-115
12. Harinarayan, V., Rajaraman, A., Ullman, J.: Implementing Data Cubes Efficiently. Proc. of ACM Sigmod Conf., Montreal, Canada (1996)
13. Kimball, R.: The data warehouse toolkit. John Wiley & Sons (1996)
14. Lin, X., Orlowska, M., Zhang, Y.: A graph-based cluster approach for vertical partitioning in database design. Data & Knowledge Engineering 11 (1993) 151-169
15. Munneke, D., Wahlstrom, K., Mohania, M.: Fragmentation of multidimensional databases. Proc. 10th Australasian Database Conf., Auckland (1999) 153-164
16. Özsu, M.T., Valduriez, P.: Principles of distributed database systems. Prentice-Hall Int. Editors (1991)
17. Papadimitriou, C.H., Steiglitz, K.: Combinatorial optimization. Prentice Hall, Englewood Cliffs (1982)
18. Raab, F. (ed.): TPC Benchmark(tm) D (Decision Support), Proposed Revision 1.0. Transaction Processing Performance Council, San Jose, CA 95112 (1995)
19. Yang, J., Karlaplem, K., Li, Q.: Algorithms for Materialized View Design in Data Warehousing Environments. Proc. 23rd Int. Conf. on Very Large Databases, Athens, Greece (1997) 136-145

Space-Efficient Data Cubes for Dynamic Environments*

Mirek Riedewald[1], Divyakant Agrawal[1], Amr El Abbadi[1], and Renato Pajarola[2]

[1] Dept. of Computer Science, Univ. of California, Santa Barbara CA 93106, USA
{mirek, agrawal, amr}@cs.ucsb.edu
[2] Dept. of Information and Computer Science, Univ. of California, Irvine CA 92697, USA
pajarola@acm.org

Abstract. Data cubes provide aggregate information to support the analysis of the contents of data warehouses and databases. An important tool to analyze data in data cubes is the range query. For range queries that summarize large regions of massive data cubes, computing the query result on-the-fly can result in non-interactive response times. To speed up range queries, values that summarize regions of the data cube are precomputed and stored. This faster response time results in more expensive updates and/or space overhead. While the emphasis is typically on low query and update costs, growing data collections increase the demand for space-efficient approaches. In this paper two techniques are presented that have the same update and query costs as earlier approaches, without introducing any space overhead.

1 Introduction

Data cubes are powerful tools to support the analysis of the contents of data warehouses and databases. A data cube is similar to a multidimensional array. Certain attributes of the database are chosen to be *measure attributes*. These are the attributes whose values are of interest to an analyst. Other attributes are selected as *dimensions* (also called *functional attributes*). The measure attributes are aggregated according to the dimensions. A *cell* of the data cube is described by a unique combination of dimension values. An example of a data cube based on the TPC-H benchmark database [9] would have the total price of an order as the measure attribute and the region of a customer and the order date as the dimensions. Such a data cube provides the aggregated total orders for all combinations of regions and dates. Queries issued by an analyst who wants to examine how the customer behavior in different regions changes over time (e.g., in order to evaluate the success of local advertising campaigns) do not need to access and join the "raw" data in the different tables. Instead the information is

* This work was partially supported by NSF grants EIA-9818320, IIS-98-17432, and IIS-99-70700.

Y. Kambayashi, M. Mohania, and A M. Tjoa (Eds.): DaWaK 2000, LNCS 1874, pp. 24–33, 2000.
© Springer-Verlag Berlin Heidelberg 2000

available and summarized from the data cube. Note that our data cube notion differs from the terminology used in [5]. We do not augment the data cube with pre-computed results of GROUP-BYs of subsets of the set of all dimension attributes. Thus our data cube notion corresponds to the data cube *core* in [5].

Aggregate range queries are useful analysis tools on data cubes. Such a range query aggregates the values of those cells that satisfy the range selection condition for all dimensions. For instance, a range query on our example data cube could "Find the total amount of orders in California over the last four months". Queries of this form are useful in discovering relationships between attributes in the database.

Analyzing data online is a highly interactive process. Analysts expect fast responses to their queries, ideally in the order of seconds at most. For massive data sets, however, range queries that access and aggregate on-the-fly the contents of a large number of cells, will show slow response times. To speed up those queries, the aggregates for sets of cells are pre-computed and stored in the data cube. This leads to well-known tradeoffs. Storing additional pre-computed values results in space overhead. Also, updates become more expensive when an update to a single cell triggers updates to all pre-computed values that include this cell in their aggregation. Different applications tolerate different update costs. While *what-if* scenarios and stock trading applications require fast updates, for other applications overnight batch processing of updates suffices. But even batch processing benefits from faster updates, since they reduce the size of the update window and allow for more frequent updates and shorter inaccessibility of the data. Ideally a data cube should support fast queries and fast updates at no extra storage cost.

An elegant algorithm for computing range queries that return the sum of the selected cells in data cubes is presented in [6]. We refer to it as the *Prefix Sum* technique (PS). The essential idea is to pre-compute the prefix sums of the data cube (see Fig. 2), which are used to answer ad hoc queries in constant time. Since the prefix sums replace the original values in the cells, the PS technique does not require additional space. The approach is mainly hampered by its update costs. In the worst case an update to a single cell requires recomputing the whole array, which is of the same size as the original data cube.

To reduce the high update costs, while still guaranteeing a constant query cost, the *Relative Prefix Sum* technique (RPS) [3] controls the cascading updates. This comes at the cost of a space overhead. In contrast, the *Hierarchical Cubes* techniques (HC) [1] do not require additional space. HC generalize the idea of RPS by allowing different tradeoffs between update and query cost. The tradeoff is selected by setting parameters that control the generation of the pre-computed values. Consequently the query and update costs depend on those parameters as well as the dimensionality and the size of the data cube. This makes a general comparison of HC to the other techniques difficult. For instance, while for some data cubes one of the HC techniques might provide a parameter setting that leads to a better query and update behavior than RPS, for other data cubes this is not the case.

The only technique that guarantees that query and update cost are both sublinear in the domain size of the dimensions for any data cube is the *Dynamic Data Cube* (DDC) [2]. The space overhead of this technique, however, is significant.

For massive data sets the space requirements of a technique become a decisive factor. Space overhead not only leads to extra costs for storage devices, but also causes additional propagations of updates and longer access times on the physical devices.

In this paper we present two new space-efficient data cube techniques – SRPS and SDDC – based on RPS and DDC, respectively. Both techniques inherit the update and query costs of their predecessors, but considerably reduce the space requirements. More precisely, they have the same storage requirement as the original data cube, thus do not introduce any space overhead. They are suitable for data warehousing environments, especially decision support and OLAP applications. SRPS efficiently supports applications where queries dominate. SDDC balances the costs of queries and updates. Thus it is especially appropriate in settings with frequent updates and enables users to analyze what-if scenarios.

In Sect. 2 we describe the SRPS and SDDC techniques. Both techniques are compared to their predecessors RPS and DDC, respectively. Section 3 concludes this article.

2 The SRPS and SDDC Techniques

In this section SRPS and SDDC are presented and compared to RPS and DDC, respectively. The following notation will be used. Let A be a data cube of dimensionality d, and let $c = [c_1, \ldots, c_d]$ be a cell that contains the value $A[c]$. Without loss of generality let the domain of each dimension attribute i be $\{0, 1, \ldots, n-1\}$. $e : f$ is a *region* of the data cube, more precisely the set of all cells c that satisfy $e_i \leq c_i \leq f_i$ for all $1 \leq i \leq d$ (i.e., $e : f$ is a hyper-rectangular region of the data cube). Cell e is the *anchor* and cell f the *endpoint* of the region. Consequently the entire data cube is anchored at $[0, \ldots, 0]$ and ends at $[n-1, \ldots, n-1]$. The set of the values in region $e : f$ is denoted $A[e] : A[f]$, and $\texttt{op}(A[e] : A[f])$ is the result of applying the aggregate operator `op` to those values.

SRPS and SDDC make use of the *inverse property* of some aggregation operators. They can be applied to any operator $\oplus$ for which there exists an inverse operator $\ominus$ such that $(a \oplus b) \ominus b = a$ (e.g., `SUM`, `COUNT`). For the SQL operator `SUM` (sum of the values of the selected cells) each region's sum can be obtained by adding and subtracting sums for appropriate regions that are anchored at $[0, \ldots, 0]$. We will refer to a region that is anchored at $[0, \ldots, 0]$ as a *prefix region*; a query that selects such a region is a *prefix query*. Note that according to [6] any range sum can be computed by combining the range sums of up to 2^d (which is a constant) prefix regions. Thus the problem of computing the sum for an arbitrary range is reduced to the problem of efficiently computing prefix sum queries. We will therefore only describe how SRPS and SDDC solve this problem.

The SRPS and SDDC techniques are described for the aggregate operator `SUM`. Other operators for which there exists an inverse operator can be handled similarly. In our analysis query and update costs are expressed in terms of the number of accessed cells of the data cube. The storage cost is measured in terms of cells as well.

2.1 RPS: The Relative Prefix Sum Technique

In this section we give an overview of the RPS technique [3]. A detailed correct analysis for RPS for high-dimensional data cubes can be found in [4]. Like the Prefix Sum technique, RPS reduces the problem of summarizing any possible range to the problem of summarizing and combining prefix regions. The main idea of RPS is to avoid the cascading updates of the Prefix Sum technique by dividing the data cube into smaller chunks of equal size, called *overlay boxes*. The prefix sums are computed and stored relative to the anchor cell of an overlay box. The array with those relative prefix sums has the same size as the original data cube. Since the relative prefix sums only provide aggregate information about the cells *inside* the overlay box, an additional data structure – the *overlay array* – is used. The overlay array provides sums for regions of cells *outside* the overlay boxes. Together the overlay and the relative prefix sum array guarantee a worst case cost of 2^d for prefix queries, a worst case cost of 2^{2d} for general range sum queries, and a worst case update cost of $(2\sqrt{n} - 2)^d$. Compared to directly storing the original data cube, the RPS technique incurs a space overhead of the size of the overlay array. Depending on the parameters (dimensionality, size of the data cube and the overlay boxes) this overhead ranges from a few percent up to almost 100% of the data cube size in some settings.

2.2 SRPS: The Space-Efficient Relative Prefix Sum Technique

Like the Relative Prefix Sum method, SRPS provides constant-time queries with an update complexity of $O(n^{d/2})$, compared to an update cost of $O(n^d)$ for the Prefix Sum technique. SRPS improves on RPS by not incurring the additional overlay array and thus removing the space overhead.

Description of the Technique. The data cube is completely partitioned into a set of disjoint hyper-rectangles of equal size. We will refer to those hyper-rectangles as *boxes*. For clarity and without loss of generality let the length of a box in each dimension be k.

Let B be a box that is anchored at cell $a = [a_1, \ldots, a_d]$. Then box B contains all cells c that satisfy $a_i \leq c_i < a_i + k$ for all $1 \leq i \leq d$. The box cells on the "upper left" surfaces, i.e., all those cells that agree with the anchor cell a in at least one coordinate, are referred to as *border cells*. The other cells, i.e., those cells c with $a_i + 1 \leq c_i < a_i + k$ for all $1 \leq i \leq d$, are *inner cells*. Essentially inner cells only store sums local to the box, while border cells include cells from outside the box into their aggregation.

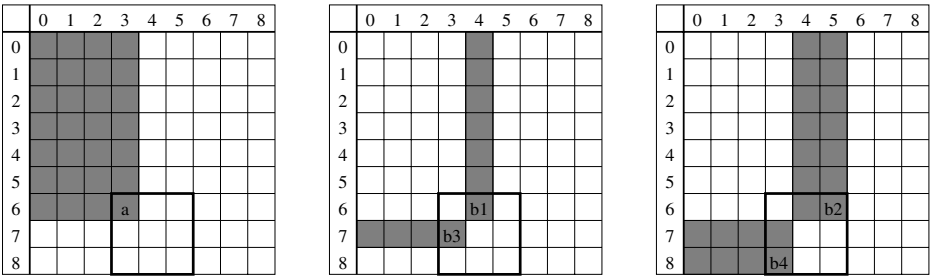


Fig. 1. Computation of border values as the sum of the values of the cells in the shaded area on array A

A	0	1	2	3	4	5	6	7	8
0	3	5	1	2	2	4	6	3	3
1	7	3	2	6	8	7	1	2	4
2	2	4	2	3	3	3	4	5	7
3	3	2	1	5	3	5	2	8	2
4	4	2	1	3	3	4	7	1	3
5	2	3	3	6	1	8	5	1	1
6	4	5	2	7	1	9	3	3	4
7	2	4	2	2	3	1	9	1	3
8	5	4	3	1	3	2	1	9	6

Original data cube

P	0	1	2	3	4	5	6	7	8
0	3	8	9	11	13	17	23	26	29
1	10	18	21	29	39	50	57	62	69
2	12	24	29	40	53	67	78	88	102
3	15	29	35	51	67	86	99	117	133
4	19	35	42	61	80	103	123	142	161
5	21	40	50	75	95	126	151	171	191
6	25	49	61	93	114	154	182	205	229
7	27	55	69	103	127	168	205	229	256
8	32	64	81	116	143	186	224	257	290

Cumulative cube for PS

	0	1	2	3	4	5	6	7	8
0	3	5	6	11	2	6	23	3	6
1	7	3	5	18	8	15	34	2	6
2	9	7	11	29	11	21	55	7	18
3	15	14	20	51	16	35	99	18	34
4	4	2	3	10	3	7	24	1	4
5	6	5	9	24	4	16	52	2	6
6	25	24	36	93	21	61	182	23	47
7	2	4	6	10	3	4	23	1	4
8	7	8	13	23	6	9	42	10	19

SRPS cube (3x3 boxes)

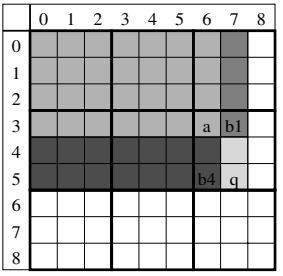
Fig. 2. Prefix Sum compared to SRPS

Any cell c in box B stores the value $\texttt{SUM}(A[l_1, l_2, \ldots, l_d] : A[c])$ where

$$\forall i : 1 \leq i \leq d \quad \begin{cases} l_i = 0 & \text{, if} \quad c_i = a_i \\ l_i = a_i + 1 & \text{, if} \quad a_i + 1 \leq c_i < a_i + k \end{cases} .$$

Note, that border cells for at least one dimension i satisfy $c_i = a_i$, while for inner cells the second inequality $(a_i + 1 \leq c_i < a_i + k)$ holds for all dimensions. We will use the term *aggregation region* for the described regions of cells. Border cells aggregate hyper-rectangular regions of cells that stretch from the surface of the box to the corresponding surface of the data cube. Inner cells c store $\texttt{SUM}(A[a_1 + 1, a_2 + 1, \ldots, a_d + 1] : A[c])$, which is the prefix sum relative to cell $[a_1 + 1, a_2 + 1, \ldots, a_d + 1]$. Figure 1 shows aggregation regions (shaded) for border cells of a two-dimensional data cube. In the example in Fig. 2 the original data cube and the corresponding SRPS cube are shown.

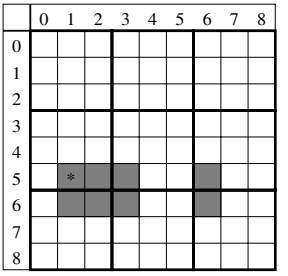
SRPS by definition does not cause any space overhead. All pre-computed values "fit" into an array of the size of the data cube. Once the SRPS cube is constructed, the original data cube can be discarded. All queries and updates are directed to the SRPS cube. The space savings compared to the RPS technique



(a) Partitioning the query region

	0	1	2	3	4	5	6	7	8
0	3	5	6	11	2	6	23	3	16
1	7	3	5	18	8	15	34	2	6
2	9	7	11	29	11	21	55	7	18
3	15	14	20	51	16	35	99	18	34
4	4	2	3	10	3	7	24	1	4
5	6	5	9	24	4	16	52	2	6
6	25	24	36	93	21	61	182	23	47
7	2	4	6	10	3	4	23	1	4
8	7	8	13	23	6	9	42	10	19

(b) Query: 99+18+52+2=171



(c) Update to [5,1]

Fig. 3. Querying and updating an SRPS cube

can be considerable. For RPS a prefix array of size n^d (i.e., size of the original data cube) and an overlay array of size $(n/k)^d(k^d - (k-1)^d) = n^d(1 - (\frac{k-1}{k})^d)$ were stored. Thus SRPS saves storage of the size $n^d(1 - (\frac{k-1}{k})^d)$.

Querying SRPS. As mentioned earlier, any range sum query can be answered by combining the results of up to 2^d appropriate prefix queries. Let $q = [q_1, \ldots, q_d]$ be the endpoint of the prefix region and $a = [a_1, \ldots, a_d]$ be the anchor of box B that contains q. Then the query region $[0, \ldots, 0] : q$ can be partitioned into non-overlapping regions which are identical to the aggregation regions of the cells in the set $\{c = [c_1, \ldots, c_d] | \forall i : 1 \leq i \leq d \wedge c_i \in \{a_i; q_i\}\}$. This set contains at most 2^d cells. Intuitively they are obtained as the "projection" of cell q to the surfaces of box B that contain the border cells (including cell q itself). Details about the partitioning are provided in [8]. Note, that the result for a prefix query can be obtained by adding values from a single box, which results in a high locality of accesses. Since a prefix query can be answered at a cost of 2^d, the *overall worst case range query cost* for SRPS becomes $2^d * 2^d = 2^{2d}$. Hence the query cost is constant irrespective of n, the size of the dimension domains of the data cube. Figure 3(a) shows an example for the partitioning of the query region for a two-dimensional data cube. The shaded cells in Fig. 3(b) need to be accessed in order to compute the prefix range sum.

Updating SRPS. In general an update to a single cell affects all those cells that store a pre-computed value that depends on that cell. Figure 3(c) shows an example. An update to cell $(5, 1)$ (marked with *) has to be propagated to each of the shaded cells.

To keep the description simple, we assume that k, the side-length of each box, evenly divides n, the side-length of the data cube. Clearly the number of cells that are affected by an update to a cell u is equal to the number of aggregation regions that contain u. From the definition of the aggregation regions it follows that at most $(n/k + k - 2)^d$ aggregation regions contain cell u. This bound is tight; it is met when cell $u = [1, 1, \ldots, 1]$ is updated. Note, that the aggregation

regions that contain the updated cell are well defined. The cells that need to be updated are the endpoints of those regions. Details of the analysis are not provided here due to space limitations and can be found in [8].

The update costs are minimal for $k = \sqrt{n}$, resulting in a *worst case update cost* of $(2\sqrt{n} - 2)^d = \mathrm{O}(n^{d/2})$. Changing k does not affect the worst case query costs.[1] Consequently, choosing $k = \sqrt{n}$ results in the optimal SRPS cube.

2.3 DDC: The Dynamic Data Cube Technique

In this section an overview of the Dynamic Data Cube technique [2] is given. As in PS, RPS, and SRPS the answer to an arbitrary range sum query is obtained by combining the results of the corresponding prefix queries.

The basic DDC technique makes use of non-intersecting boxes which store pre-computed values that only summarize the cells in the box. Those values are stored in the "lower right" surfaces of the box (border cells) and summarize the cells in a region that has the anchor of the box as its anchor and the surface cell as the endpoint. The boxes are organized into a tree that recursively partitions the original data cube. The root node encompasses the entire data cube. It forms children by dividing its range in each dimension in half. Each of the children are in turn subdivided into children, and so on.

The values in the border cells are cumulative. Thus an update to the anchor cell of a box has to be propagated to all border cells in the box. To reduce the update cost [2] introduces the B^c tree. B^c trees are standard B-trees whose non-leaf nodes are augmented by an auxiliary value that stores the sum of the leafs in the left sub-tree. By taking advantage of these auxiliary values, B^c trees provide balanced query and update costs of $\log m$ for any one-dimensional array of size m that stores cumulative values. By storing the (one-dimensional) border cell arrays in B^c trees, update and query costs of $\mathrm{O}(\log^2 n)$ can be achieved for two-dimensional data cubes. For data cubes with $d > 2$ dimensions the $(d-1)$-dimensional surfaces that contain the border cells are recursively stored as $(d-1)$-dimensional data cubes. Thus DDC with B^c trees and the recursive technique for storing the border values guarantees the polylogarithmic update and query costs of $\mathrm{O}(\log^d n)$.

2.4 SDDC: The Space-Efficient Dynamic Data Cube Technique

Like the Dynamic Data Cube, SDDC balances query and update costs to $\mathrm{O}(\log^d n)$. Those bounds are maintained at much lower storage costs. For clarity, we will first describe a simpler basic approach that has higher update costs, and then SDDC with the polylogarithmic costs.

[1] The only exception occurs for $k = 1$, when SRPS collapses to the Prefix Sum technique.

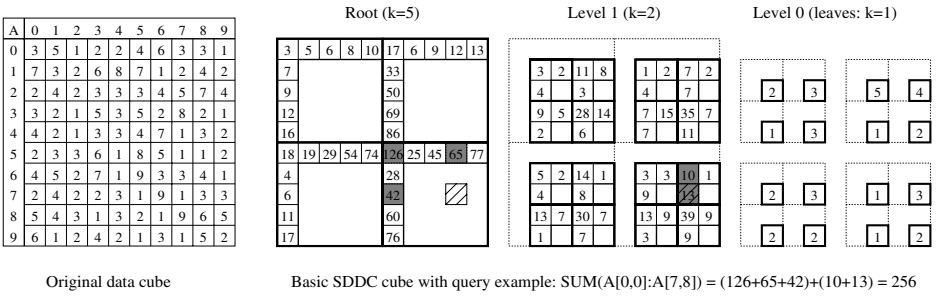


Fig. 4. Original data cube and corresponding SDDC cube

The Basic SDDC. To construct the SDDC cube, the data cube is first partitioned into boxes using the same technique as SRPS, except for two differences. First, the side-length of a box is set to $k = n/2$, i.e., the data cube is partitioned into 2^d boxes of equal size. Second, while the aggregation regions of the border cells remain the same, the inner cells do not store relative prefix sums any more. Instead a recursive approach is taken. For each box the region that contains all inner cells of that box (which is a hyper-cube of side-length $n/2 - 1$) is partitioned into 2^d non-intersecting boxes of equal size. Their regions of inner cells are then in turn partitioned into 2^d boxes, and so on. Conceptually the boxes of the basic SDDC form a tree where each node corresponds to a box. The root node encompasses the entire data cube. The children of a node are those smaller boxes that partition the regions of the inner cells of the node. They store the corresponding border cell values. At the leaf level nodes simply store the value of the single cell that corresponds to the node. Since the side-length of a node is less than half the side-length of its parent, the tree height can not exceed $\log_2 n$. Figure 4 presents a data cube and the corresponding basic SDDC.

Instead of dividing the region of the inner cells in each dimension in half, one could alternatively choose other partitionings into boxes. Partitions with flexible split positions in the different dimensions can be used to identify similarity regions, which could be exploited by operations that can take advantage of low variance distributions (i.e., for data compression).

Due to how the boxes are created, the complete basic SDDC fits into the space of the original data cube (with n^d cells). The storage savings compared to the basic DDC approach are considerable. In [8] we show that the basic DDC requires more than twice the space of the original data cube. Thus our new basic SDDC technique reduces the space overhead by more than the size of the original data cube!

To find the sum for any prefix region, the tree is descended and the appropriate border values are added. On a tree level the query is answered as described for SRPS. The only difference is that instead of accessing an inner cell, the query recursively accesses the corresponding child node (see [8] for details). In the example in Fig. 4 the cells that are accessed in order to compute $\texttt{SUM}(A[0,0] : A[7,8])$ are shaded; cell $[7,8]$ is hatched. Since the partitioning of

the data cube is non-overlapping, at most one box per level contains the endpoint of the query region. For each such box at most $2^d - 1$ border cells have to be accessed (analysis identical to SRPS). Since the tree has at most $\log_2 n$ levels, the cost for any prefix query is less or equal than $(2^d - 1)\log_2 n = \mathrm{O}(\log n)$.

Updates on the basic SDDC are very expensive in the worst case. Consider an update to cell $[1, 1, \ldots, 1]$. This update already affects $\mathrm{O}(n^{d-1})$ border cells in the root node (see [8] for details).

SDDC with Improved Updates. The problem the basic SDDC faces regarding updates is similar to the update problem for the basic DDC technique. To reduce the update costs, we can apply the same technique as for DDC, i.e., using $\mathrm{B^c}$ trees for balanced update and query costs on two-dimensional data cubes, and storing the border values of higher-dimensional data cubes recursively (see Sect. 2.3). However, $\mathrm{B^c}$ trees and the recursive approach introduce unnecessary redundancy. We follow a similar approach, but remove the additional storage requirements.

Recall that the values of the border cells in the same surface are cumulative, which results in the high worst case update costs. To reduce the costs for one-dimensional arrays of border cells we use an elegant technique that embeds a tree into the array. The main idea is to first replace the cumulative values by the corresponding differences of the values of neighboring cells and then to apply the basic SDDC technique to this array of differences. Queries and updates are processed as described for the basic SDDC technique, resulting in a worst case cost of $\log n$ for both operations. Thus the $\mathrm{B^c}$ tree is replaced by a data structure which does not add any space overhead compared to storing the original array.

The DDC technique stores $(d-1)$-dimensional surfaces of border cells recursively as $(d-1)$-dimensional data cubes. Since the surfaces are overlapping, redundancy is introduced. SDDC removes this redundancy by ensuring that values in the overlapping regions are stored only once. The idea is to embed the recursively computed values into the space of exactly those values they replace.

Note that improving on the $\mathrm{B^c}$ trees and the recursive technique for storing the values of the border cells further increases the space savings of SDDC compared to DDC. Due to the improvements, SDDC has the *same storage consumption as the original data cube*. Its query and update costs are $\mathrm{O}(\log^d n)$, which is sublinear in the side-length of the data cube. A more detailed description of the technique can be found in [8].

3 Conclusion

Aggregate range queries are useful tools for analyzing information that is stored in data cubes. For massive data sets, however, accessing and aggregating the relevant data on-the-fly can result in slow responses that negatively affect the analysis process. In this paper two new techniques were discussed that speed up range queries by storing pre-aggregated information, while still supporting

efficient updates. Using SRPS or SDDC cubes instead of the original data cube provides efficient queries and updates without introducing any space overhead.

To be more precise, we developed one technique (SRPS) that guarantees that any aggregate range query is answered in constant time and that no update results in costs higher than the square root of the data cube size. We presented another technique (SDDC), that improved the only existing technique which provides provably polylogarithmic worst case query and update costs for any data cube. Our technique guarantees the same query and update costs, while reducing the space overhead by an amount of space that is greater than the size of the original data cube.

Thus our new techniques efficiently support online aggregation for massive data sets. Reducing the space requirements not only saves storage costs, but at the same time reduces the real access and update times. This is not reflected in our cost formulas that are only based on cell accesses. Real access costs, however, also depend on cache sizes and I/O times for external storage devices. For real query and update costs, smaller space consumption can be very beneficial. Similar to all methods that are based on prefix sums, our approaches are particularly suited for dense data sets. We are currently developing techniques for sparse high-dimensional data cubes [7]. Also, to compare our techniques to previous approaches in more detail, further theoretical and experimental evaluations are needed.

References

[1] C.-Y. Chan and Y. E. Ioannidis. Hierarchical cubes for range-sum queries. In *Proc. Int. Conf. on Very Large Databases (VLDB)*, pages 675–686, 1999.

[2] S. Geffner, D. Agrawal, and A. El Abbadi. The dynamic data cube. In *Proc. Int. Conf. on Extending Database Technology (EDBT)*, pages 237–253, 2000.

[3] S. Geffner, D. Agrawal, A. El Abbadi, and T. Smith. Relative prefix sums: An efficient approach for querying dynamic OLAP data cubes. In *Proc. Int. Conf. on Data Engineering (ICDE)*, pages 328–335, 1999.

[4] S. Geffner, M. Riedewald, D. Agrawal, and A. El Abbadi. Data cubes in dynamic environments. *Data Engineering Bulletin*, 22(4):31–40, 1999.

[5] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Data Mining and Knowledge Discovery*, pages 29–53, 1997.

[6] C. Ho, R. Agrawal, N. Megiddo, and R. Srikant. Range queries in OLAP data cubes. In *Proc. Int. Conf. on Management of Data (SIGMOD)*, pages 73–88, 1997.

[7] M. Riedewald, D. Agrawal, and A. El Abbadi. pCube: Update-efficient online aggregation with progressive feedback and error bounds. In *Proc. Int. Conf. on Scientific and Statistical Database Management (SSDBM)*, 2000. To appear.

[8] M. Riedewald, D. Agrawal, A. El Abbadi, and R. Pajarola. Space-efficient data cubes for dynamic environments. Technical Report TRCS00-05, UC Santa Barbara, 2000.

[9] Transaction Processing Performance Council. TPC-H benchmark (1.1.0). Available at http://www.tpc.org.

On Making Data Warehouses Active

Michael Schrefl[1] and Thomas Thalhammer[2]

[1] University of South Australia, Advanced Computing Research Center
schrefl@cs.unisa.edu.au
[2] University of Linz, Data and Knowledge Engineering
thalhammer@dke.uni-linz.ac.at

Abstract. Data warehouses, which are the core elements of On-Line Analytical Processing (OLAP) systems, are *passive* since all tasks related to analyzing and making decisions must be carried out *manually*. This paper introduces a novel architecture, the *active data warehouse*, which applies the idea of event-condition-action rules (ECA rules) from active database systems to automize repetitive analysis and decision tasks in data warehouses. The work of an analyst is mimicked by *analysis rules*, which extend the capabilities of conventional ECA rules in order to support multidimensional analysis and decision making.

1 Introduction

Database systems are the core elements of information systems that support On-Line Transaction Processing (OLTP) in day-to-day business. Originally, every action in a (passive) database system had to be initiated by explicit user interaction or application request. Event-condition-action (ECA) rules were introduced to extend (passive) database systems with reactive capabilities. An ECA rule relys on the basic idea that when a specified happening occurs (the *event*) and a certain boolean expression evaluates to "TRUE" (the *condition*) then an appropriate *action* will be carried out automatically without explicit user interaction or application request. A database system that supports ECA rules is referred to as *active* database system.

Data warehouses, which are the core elementes of On-Line Analytical Processing (OLAP) systems [6], are *passive*. Although most tasks related to extracting, transforming, and loading data from various operational OLTP systems can be automized [5], all tasks related to analyzing data and making decisions must be carried out manually. In decision making, analysts carry out different kinds of decision tasks. In the case of *routine decision tasks*, analysts apply well established decision criteria on routine problems through standardized analyses that are carried out on a day-to-day basis. In the case of *non-routine decision tasks*, analyses are carried out on the basis of "trial-and-error", since decision criteria are vague and cannot be formalized. Finally, *semi-routine decision tasks* contain elements of routine decision tasks and non-routine decision tasks. We employ techniques from active database systems to automize routine decision tasks and the routinizable elements of semi-routine decision tasks in data warehouses. The resulting architecture is called *active data warehouse* (cf. Fig. 1).

Y. Kambayashi, M. Mohania, and A M. Tjoa (Eds.): DaWaK 2000, LNCS 1874, pp. 34–46, 2000.
© Springer-Verlag Berlin Heidelberg 2000

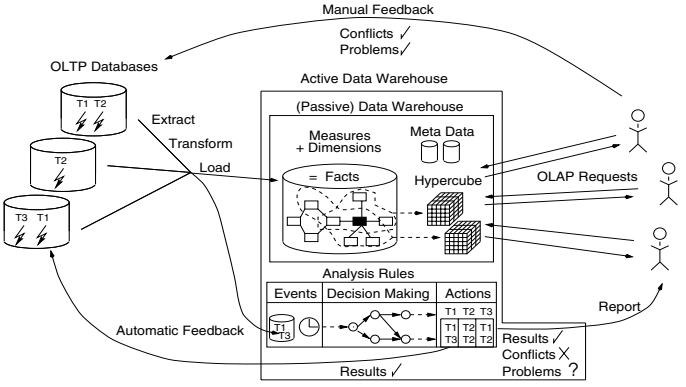


Fig. 1. Architecture of an Active Data Warehouse

In analyzing data for a routine decision task (e.g., change the price of an article), analysts follow a "top down" approach by initially inspecting a multi-dimensional cube on very coarse dimension levels (e.g., inspect the *annual* sales figures of the article). The outcome of analyzing the cube is (1) the decision to perform the task, or (2) the decision not to perform the task, or (3) the decision to continue analysis on finer levels of granularity (e.g., sales figures are inspected on a *monthly* basis). To enable automatic decision making in an active data warehouse, we introduce a data structure – the *analysis graph* – which represents all cubes that are required to analyse a certain business problem. For each cube there are three conditions that represent the choices in making decisions. To automize analysis and decsion making for routine decision tasks, we introduce *analysis rules* as an extension to ECA rules from active database systems. We introduce an event model and an action model to extend data warehouses with reactive capabilities. The event of an analysis rule is a timepoint, at which the rule has to be fired. An analysis rule's action is the decision to execute or not to execute a transaction in an OLTP system. Analysis rules realize the process of decision making by evaluating the conditions associated with a cube of the analysis graph in a "top down" fashion.

The remainder of this paper is organized as follows. In Section 2, we present an event model and an action model for active data warehouses. Section 3 introduces the analysis graph. Section 4 introduces analysis rules and a model for decision making. Finally, Section 5 summarizes our achievements.

2 Events and Actions

In this paper, we view data in an active data warehouse from an object-oriented perspective, which is independent from a certain implementation (e.g., ROLAP, MOLAP). A basic assumption is that the structure of an object type (i.e., its attributes, relationships) in an OLTP database will be represented by a dimension level and a level description of one of the data warehouse's dimensions (as described in [7]). As we will see, the object-oriented view is necessary to extend the architecture of a conventional data warehouse with an action model and an

event model. In the remainder of this section, we describe the event model and the action model from an object-oriented view.

2.1 Event Model

An active data warehouse uses events to determine the *timepoints*, at which analysis and decision tasks are initiated automatically. These timepoints can be either known in advance (e.g., "at the end of every quarter ...") or as a reaction to recent happenings in the data warehouse's source systems (e.g., "twenty days after the price of an article was changed ..."). We distinguish between *event classes* and *event types*. *Event classes* collect instances of event types. *Event types* define the structure of their instances, providing at least an occurrence time and a unique event identifier for each instance. An instance of an event type is called *event*. Due to the typical temporal nature of analysis and decision tasks, active data warehouses use an event model that is more primitive than the event model of active database systems [10, 3, 4]. Our event model provides three basic event types: (1) *OLTP method events*, (2) *calendar events* and (3) *relative temporal events*.

An *OLTP method event* represents the invocation of a method in an OLTP database. We extend the data warehouse's import mechanism to make OLTP method events available in the data warehouse. OLTP method events serve as a "reference point" to relative temporal events, which are used to initiate analyses.

Calendar events represent fixed points in time, at which analysis and decision processes will be initiated. Calendar classes may be populated either manually or automatically with the help of a calendar tool.

A *relative temporal event* r is an event, whose occurrence depends on the occurrence of another event e, which serves as the reference point of r. The dependence on e is twofold: First, the existence of r depends on the existence of e (i.e., r occurs only if e occurred). Second, r depends temporally on e (i.e., r occurs a specified period *after* e occurred). Relative temporal events are defined and populated within the data warehouse.

2.2 Action Model

The purpose of an analysis rule is to make "decisions" for objects that are available in OLTP systems *and* in the data warehouse. Since we follow an object-oriented approach, a "decision" means to invoke or not to invoke a method on a certain object in an OLTP system. If we refer to the term "method" in the following, we mean methods of an OLTP object type that represent transactions. These methods represent the "decision space" of an active data warehouse. To make the behavior of an OLTP object type available to analysis rules, the action model of an active data warehouse includes (1) the specifications of the OLTP object type's methods together with required parameters, (2) the preconditions that have to be satisfied before the method can be invoked on an instance, and (3) a conflict resolution mechanism, which solves contradictory decisions of different analysis rules. Since an object may be subject to decisions by different

analysis rules, decision conflicts may occur. To detect such conflicts, we use a "conflict table" covering the methods of a certain OLTP object type. A conflict table consists of tupels $\langle m_1, m_2, m_3 \rangle$, where m_1 and m_2 identify two conflicting methods and m_3 specifies the conflict resolution method that will be finally executed in OLTP systems. We refer to such conflicts as *inter-rule conflicts*. If a conflict cannot be solved automatically it has to be reported to analysts for further manual treatment.

Example 1. An active data warehouse may invoke methods `withdrawFromMarket(location: COUNTRY)` and `changePrice(newPrice: DOLLAR)` for instances of object type `Article` in OLTP systems. These two methods cause a conflict if the price of an article should be changed and if the article should be withdrawn from the market at the same time. Thus, the conflict table contains the entry ⟨ `withdrawFromMarket` , `changePrice` , `withdrawFromMarket` ⟩, which specifies method `withdrawFromMarket` as the conflict resolution method.

Decision making in an active data warehouse will usually be carried out in an asynchronous manner, i.e., without having access to OLTP systems. Then, out-of-dateness of decisions may occur since OLTP systems may have changed manually during the period between the data warehouse was refreshed and decisions are exported to OLTP systems. Such problems are reported to analysts for further manual treatment together with non-routinizable analysis cases, for which an active data warehouse could not come to a definite decision. Out-of dateness of decisions can be recognized by (1) checking whether the object for which the method will be executed is still available in the OLTP system and (2) by checking whether the method's precondition still holds for the object in the OLTP system.

3 The Analysis Graph

The analysis graph represents all multidimensional cubes, which are necessary for automatic analysis and decision making. An analysis graph is specified in a "top down" fashion once by an analyst and is then used by an analysis rule. By "top down", we mean that cubes will be "derived" from existing more general cubes using OLAP operators `DRILLDOWN` and `SLICE` on existing cubes by specifying only the new dimension level for `DRILLDOWN` or an additional condition in the case of `SLICE`. This approach of specifying cubes is referred to as *incremental specification*. Although incremental specification of cubes is natural to the way an analyst thinks, each cube needs to be created internally by applying OLAP operator `ROLLUP` on the *base cube* since a more general cube does not provide cells at levels of detail needed by `DRILLDOWN`. Before defining the analysis graph, we (1) introduce a new definition of *cube* that supports incremental specification of cubes and (2) describe how analysts specify cubes incrementally using OLAP operators `DRILLDOWN` and `SLICE`.

Our data model follows a cube-oriented approach [12], which generally distinguishes between the notions of *dimension*, *dimension level*, *level description*,

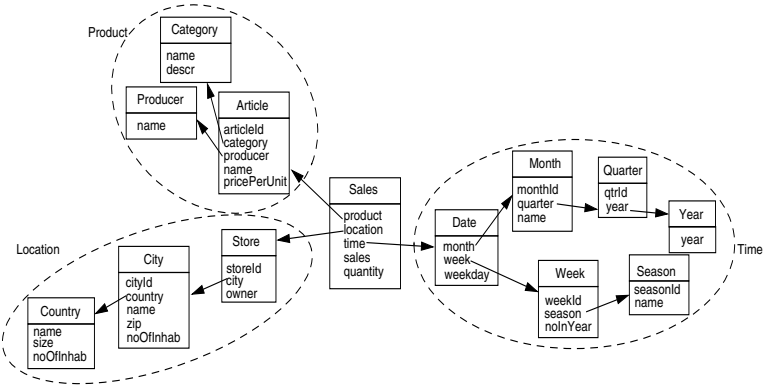


Fig. 2. Retail Data Warehouse Scheme

base cube (or *fact table*), and *cell* (or *fact*). Together with OLAP operators *Slice*, *Drilldown* and *Rollup* these concepts have been well discussed in recent years [9, 2, 1, 11] and are thus considered to be understood.

Example 2. Figure 2 represents a typical data warehouse scheme in retail trade. Base cube `Sales` provides information about dollar amounts of `sales` and the sold `quantity` according to dimensions `Product`, `Location` and `Time`. The levels of dimension `Time` constitute the two hierarchies `Date` → `Month` → `Quarter` → `Year` and `Date` → `Week` → `Season`, which are the paths for `Rollup` and `Drilldown`. Similar hierarchies exist for dimensions `Product` and `Location`. Each dimension level is described by a level description, e.g., dimension level `Location[Store]` is described by level description `Store(storeId, city, owner)`. This example serves as a running example in our paper.

Although there exists a basic agreement on fundamental multidimensional concepts, the various definitions of *cube* [8, 1, 11] are not sufficient to specify cubes incrementally. These approaches lack of one or several components a multidimensional cube c must consist of such that cubes can be specified "top down" and created "bottom up" from the base cube. These components are:

1. a cube scheme $S = (A_1 : D_1[l_1], ..., A_n : D_n[l_n], M_1 : dom(M_1), ..., M_k : dom(M_k))$, which defines the dimension attributes A_i and the measure attributes M_i of c; the domain of A_i are the values which belong to a level l_i of a certain dimension D_i, denoted as $D_i[l_i]$,
2. a base cube $b(B)$[1], from which cells are taken for aggregation,
3. a set of conditions C, which determine a subset of cells of $b(B)$ (denoted as $b(B)_C$) for aggregation,
4. a list of aggregation functions AGG, which rollup the subset of cells of $b(B)_C$, where each $f_i \in AGG$ will be evaluated for each subset of cells of $b(B)_C$

[1] B is a *base cube scheme*, whose dimension attributes A_i represent the bottom levels of the respective dimensions.

whose members rollup to the same dimension value for every dimension attribute of S.

Example 3. Suppose that cube `SalesArticlesCountriesQuarters` consists of cube scheme `(P: Product[Article], L: Location[Country], T: Time[Quarter], SalesTotal: DOLLAR)`. Attributes `P`, `L`, and `T` are dimension attributes; attribute `SalesTotal` is a measure attribute. Base cube `Sales` provides the cells for rollup using function `SUM(sales)`.

To specify cubes incrementally, we use the three OLAP operators `ROLLUP`, `DRILLDOWN`, and `SLICE`. `ROLLUP` is needed to create the "root" cube from the base cube. By applying `DRILLDOWN` or `SLICE` to the "root" cube or on a cube, which was derived from the root cube (denoted as c_{old}), a new cube c_{new}[2] will be created.

If applied to c_{old} with schema $S_{old} = (A_1 : D_1[l_1], ..., A_n : D_n[l_n], M_1 : dom(M_1), ..., M_k : dom(M_k))$, OLAP operator `DRILLDOWN` A_i `TO` $l_{i_{new}}$ defines c_{new} as follows: (1) $S_{new} = (A_1 : D_1[l_1], ..., A_{i-1} : D_{i-1}[l_{i-1}], A_i : D_i[l_{i_{new}}], A_{i+1} : D_{i+1}[l_{i+1}], ..., A_n : D_n[l_n], M_1 : dom(M_1), ..., M_k : dom(M_k))$, (2) $b(B)_{new} = b(B)_{old}$, (3) $AGG_{new} = AGG_{old}$, and (4) $C_{new} = C_{old}$.

If applied to c_{old}, OLAP operator `SLICE <cond>` defines c_{new} as follows (1) $S_{new} = S_{old}$, (2) $b(B)_{new} = b(B)_{old}$, (3) $AGG_{new} = AGG_{old}$, and (4) $C_{new} = C_{old} \cup \{$`<cond>`$\}$.

We provide a simple textual language to specify cubes incrementally.

```
DEFINE SalesArticlesCountriesQuarters AS
  (Product P, Location L, Time T, SUM(Sales) SalesTotal)
  ROLLUP P TO Article, T TO Quarter, L TO Country
  FROM   Sales;
```

(a)

```
DEFINE SalesAustraliaQuarters AS
  SLICE L.Name = 'Australia'
  FROM SalesArticlesCountriesQuarters;
```

(b)

Fig. 3. Incremental specification of cubes

Example 4. Figure 3 (a) shows the specification of cube `SalesArticlesCountriesQuarters`, which was described in the previous example. Hypercube `SalesAustraliaQuarters` (cf. Fig. 3 (b)) is derived from `SalesArticlesCountriesQuarters` by applying OLAP operator `SLICE`, which selects all cells that comply with condition `L.Name = 'Australia'`.

Before we define the analysis graph, we have to define the *more specific* relationship between two cube schemes R and S. R is said to be *more specific* than S, denoted as $R < S$, if (1) each dimension attribute A_i of R represents at least the same dimension level than the equivalent dimension attribute A_i of S and (2) there are one or more dimension attributes of R which represent a more detailed dimension level than the equivalent dimension attributes of S. If R is more specific than S, then S is more general than R (denoted as $S > R$). A cube c_{new} is said to be *more specific* than another cube c_{old}, denoted as $c_{new} < c_{old}$, iff (1) $S_{new} < S_{old}$ and $C_{new} \supseteq C_{old}$ or (2) $S_{new} = S_{old}$ and $C_{new} \supset C_{old}$.

[2] We distinguish the components of c_{old} and c_{new} by using the subscript of the respective cube, e.g., $b(B)_{old}$ is the base cube of c_{old}.

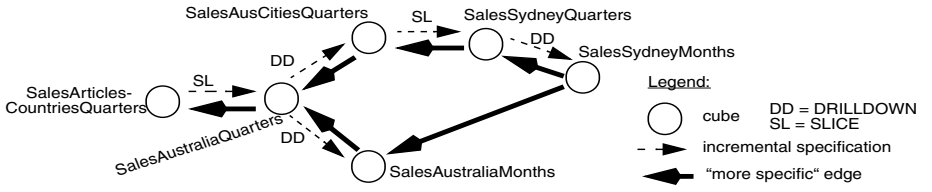


Fig. 4. Analysis graph

The *analysis graph* is a directed acyclic graph $AG = (V, E)$. V is a set of cubes. $E \subseteq V \times V$ is a set of edges, representing direct "more specific than" relationships between cubes: $E = \{(c_1, c_2) \in V \mid c_1 < c_2 \wedge \nexists c_3 \in V : c_1 < c_3 \wedge c_3 < c_2\}$. An analysis graph must have a single "root" c_r, i.e., $\forall c \in \{V \backslash \{c_r\}\} : c < c_r$. The predicate $super(c)$ denotes the set of direct more general cubes of some cube c, i.e., $\{c_j \in V \mid (c, c_j) \in E\}$.

OLAP operator `DRILLDOWN` creates a *more specific* cube that represents measures in more detail and OLAP operator `SLICE` creates a *more specific* cube that represents a subset of cells. Although analysts specify cubes incrementally by using these OLAP operators, the complete set of direct "more specific" edges of the analysis graph may be different.

Example 5. Figure 4 depicts an analysis graph, which consists of cubes `SalesArticlesCountriesQuarters`, `SalesAustraliaQuarters`, `SalesAustraliaMonths`, `SalesAusCitiesQuarters`, `SalesSydneyQuarters`, and `SalesSydneyMonths`. Hypercube `SalesArticlesCountriesQuarters` represents the "root" of the analysis graph. Dashed arrows depict how these cubes were specified incrementally by an analyst. Bold arrows depict all direct "more specific" edges between these cubes.

4 Analysis Rules

Analysis rules are templates for automatic analyses and decision making. The basic idea behind analysis rules is to extend conventional ECA rules from active database systems with multidimensional features (cubes, OLAP operators) of data warehouses and an approach for decision making.

4.1 Specifying Analysis Rules

The specification of an analysis rule consists of (1) the *primary dimension level*, which determines the OLTP object type whose instances are subject to decision making, (2) the rule's *condition*, which selects the objects of the primary dimension level for further rule processing, (3) the *event class*, which triggers the rule if a new event of this class occurs, (4) the *analysis graph*, whose nodes constitute the cubes for analysis, (5) a *decision step* for each cube of the analysis graph representing the rule's conditions in decision making, and (6) the *action*, which represents the rule's operational task.

```
DEFINE ANALYSIS RULE HighPricedSoftDrinksQtrAusAR

FOR    P:Product[Article]
WHERE (P.category = 'SoftDrinks') AND (P.pricePerUnit > 10)
ON     EndOfQuarter eoq

USING CUBES

   SalesAustraliaThisQuarter AS (Product P, Location L, Time T, SUM(Sales) SalesTotal)
      SLICE L.Name = 'Australia' AND T.qtrId = eoq.quarterId
      FROM SalesArticlesCountriesQuarters;

   SalesAustraliaThisQuarterMonths AS DRILLDOWN T TO Month
      FROM SalesAustraliaThisQuarter;

   SalesAusCitiesThisQuarter AS DRILLDOWN L TO Country
      FROM SalesAustraliaThisQuarter;

   SalesSydneyThisQuarter AS SLICE L.name = 'Sydney'
      FROM SalesAusCitiesThisQuarter;

   SalesSydneyThisQuarterMonths AS DRILLDOWN T TO Month
      FROM SalesSydneyThisQuarter;

ANALYZE SalesAustraliaThisQuarter
        TRIGGER ACTION IF SalesTotal < 100,000
        DETAIL ANALYSIS IF SalesTotal < 500,000;

ANALYZE SalesAustraliaThisQuarterMonths
        TRIGGER ACTION IF TREND(s1.SalesTotal, s2.SalesTotal, s3.SalesTotal) = 'DOWN'
                       FOR CELLS s1, s2, s3
                       WHERE s1.T = eoq.1stMon AND s2.T = eoq.2ndMon AND s3.T = eoq.3rdMon;

ANALYZE SalesAusCitiesThisQuarter
        TRIGGER ACTION IF AVG(SalesTotal) < 15,000 GROUPED BY T
        DETAIL ANALYSIS IF AVG(SalesTotal) < 30,000 GROUPED BY T;

ANALYZE SalesSydneyThisQuarterMonths
        TRIGGER ACTION IF ((s2.SalesTotal - s1.SalesTotal)/s2.SalesTotal)*100 = -10
                       FOR CELLS s1, s2
                       WHERE s1.T = eoq.1stMon AND s2.T = eoq.2ndMon;

TO EXECUTE P.withdrawFromMarket('Australia')

END HighPricedSoftDrinksQtrAusAR
```

Fig. 5. Analysis Rule `HighPricedSoftDrinksQtrAusAR`

As described in Sect. 2.2, the decision of an analysis rule is to invoke or not to invoke a method on the objects of a OLTP object type in an OLTP system. Such objects are represented as values of one of the data warehouse's dimensions. This dimension is referred to as the *primary dimension*. The dimension level whose values describe the instances of this object type is called *primary dimension level*. If we compare analysis rules with conventional ECA rules in active object-oriented database systems, then the primary dimension level represents the object type of pseudo-variable "self".

An analysis rule's action part refers to one of the primary dimension level's OLTP methods that are defined for the OLTP object type and are included in the action model of the active data warehouse.

Example 6. The primary dimension level of analysis rule `HighPricedSoftDrinksQtrAusAR` of Fig. 5 is specified with clause `FOR P:Product[Article]`. In its action part, this analysis rule refers to OLTP

method `withdrawFromMarket(...)`, which is defined for OLTP object type `Article`.

Several analysis rules may share the same OLTP method as their action. An analysis rule's condition is used to classify the entities of the primary dimension level to be analyzed by different analysis rules, which all refer to the same OLTP method as their action. The condition-part of an analysis rule is a simple boolean predicate that refers to describing attributes of the primary dimension level.

Example 7. In our retail example, high priced "soft drinks" (price per unit $>$ \$ 10) are analyzed in a different way and at different time points than low priced "soft drinks" (price per unit $<$ \$ 10). For this purpose, two analysis rules `HighPricedSoftDrinksQtrAusAR` and `LowPricedSoftDrinksMonAusAR` are specified, which both refer to `withdrawFromMarket(...)` as their action.

The event-part of an analysis rule refers to an event class of the active data warehouse's event model, which may be a *calendar class* or a *relative temporal event class* (cf. Sect. 2.1). In the case of a calendar class, the analysis rule will be fired for all objects of the primary dimension level[3]. In the case of a relative temporal event class, the analysis rule will be fired exclusively for the object of the OLTP event, which serves as reference point to the relative temporal event. The set of objects, for which an analysis rule is fired and which comply with the analysis rule's condition is referred to as *analysis scope*.

Example 8. Analysis rule `HighPricedSoftDrinksQtrAusAR` of Fig. 5 will be fired for all entities of the primary dimension level `Product[Article]` at the occurrence of an event of type `EndOfQuarter`.

Since an analysis rule is specified for a certain primary dimension level, the cells of each cube of the analysis graph must describe the instances of the primary dimension level. As a consequence, analysts cannot apply OLAP operators `ROLLUP`, `DRILLDOWN`, and `SLICE` on a cube's primary dimension level. The remaining non-primary dimension levels are referred to as *analysis dimensions*.

In specifying the cubes of the analysis graph, "inner cubes" will be defined as local components of the analysis rule, whereas the analysis graph's "root" cube is a globally available cube, which can be "imported" by the analysis rule. The reasons for this separation are the following:

First, the definition of a cube may depend on the event of the analysis rule, e.g., the event may be used by a `SLICE` condition. Second, the cubes of an analysis graph are closely related to the rule's decision criteria, which reduces their reusability for analyses by other analysis rules. Third, changes in the specification of individual cubes will affect a single analysis rule only. There are many rules concerning the entities of a certain dimension level. Locality helps here to eliminate unwanted side-effects to other analysis rules. Fourth, some general cubes should be available to several analysis rules as "starting points" to reduce redundancies.

[3] Notice that an efficient implementation will execute these rules in a set-oriented manner.

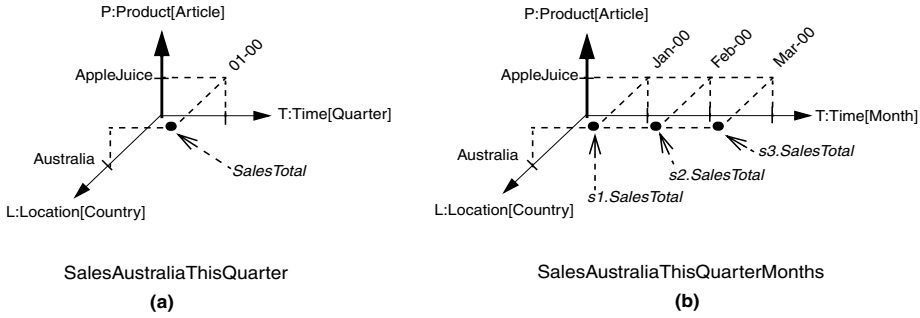


Fig. 6. Identifying cells

Example 9. Local cube `SalesAustraliaThisQuarter` is specified by applying OLAP operator `SLICE L.name = 'Australia' AND T.qtrId = eoq.quarterId` on global cube `SalesArticlesCountriesQuarters`.

After analyzing an instance of the primary dimension level using a certain cube of the analysis graph, there are three choices available: (a) **execute** the rule's action for the instance of the primary dimension level, (b) **don't execute** the rule's action for the instance of the primary dimension level, and (c) **continue analysis** for the instance on a more specific cube. In automatic analysis and decision making, our approach comprises choices (a) and (c), which will be represented as boolean conditions referring to the cells of a cube. The complement of these two conditions is (b), which represents the decision not to execute the rule's action in the OLTP system. In the following, we refer to condition (a) as *triggerActionCond* and condition (c) as *detailAnalysisCond*. The triplet $\langle cube, triggerActionCond, detailAnalysisCond \rangle$ is called *decision step*. Each cube of the analysis graph is analysed by a single decision step. Syntactically, a decision step is specified by `ANALYZE` *cube* `TRIGGER ACTION IF` *triggerActionCond* `DETAIL ANALYSIS IF` *detailAnalysisCond*. The predicates of the two conditions refer to the cells of *cube*.

Each cube describes the entities of the primary dimension level by a single cell or by a sets of cells. If an entity of the primary dimension level is described by a single cell, the decision step's conditions (*triggerActionCond* and *detailAnalysisCond*) can refer to this cell directly.

Example 10. Figure 6 (a) illustrates that each entity of the primary dimension level `Product[Article]` (depicted by a bold arc) of cube `SalesAustraliaThisQuarter` represents a single cell, since slice conditions `Country.Name = 'Australia'` and `Quarter.qtrId = eoq.quarterId` rollup to a single dimension value for a given article. Thus, the conditions of the first decision step in Fig. 5 refer directly to the cells of this cube.

If an entity of the primary dimension level is described by a *set of cells*, then least one analysis dimension level describes the primary dimension level with more than one dimension value. In this case, the decision step's conditions refer to individual cells by identifying each single cell separately using *cell identifiers*.

Identifying individual cells will be accomplished by using an SQL like query, which consists of two components:

1. The *cells-part* resembles the `FROM` clause in SQL. It defines alias names (*cell identifiers*) for the cube of the decision step. All cell identifiers belong to the same entity of the primary dimension level, but each cell identifier refers to a different cell for analysis. Syntactically, the cells-part is specified by `FOR CELLS <cell-id 1>, <cell-id 2>, ....`
2. The *selection-part* represents the selection criterions for these cell identifiers. It is similar to the `WHERE` clause in SQL. A selection criterion is a boolean predicate, whose operands refer to the attributes of the level descriptions of a cell's dimension levels or to atomic values. Valid operators are $<$, $>$, $\leq$, $\geq$, $=$, and negations. Syntactically, the selection-part is specified by `WHERE <selection criteria>`.

Example 11. As illustrated in Fig. 6, the decision step specified for cube `SalesAustraliaThisQuarterMonths` uses cell identifiers `s1`, `s2`, and `s3` to indentify the sales of the first, second, and third month of a quarter for an instance of the primary dimension level `Product[Article]`.

If the conditions *triggerActionCond* or *detailAnalysisCond* need to refer to subtotals of measures of cells describing the same entity of the primary dimension level, an optional *aggregation-part* offers the possibility to aggregate these cells along several analysis dimensions. The aggregation-part is similar to the `GROUP BY` clause in SQL, except that it aggregates the cells describing the same primary dimension level implicitly. Thus, only aggregation along analysis dimensions must be specified explicitly using clause `GROUPED BY <analysis dimension levels>` and aggregation functions, such as `SUM`, `COUNT`, and `AVG` in the conditions.

Example 12. The analysis conditions of the decision step specified for cube `SalesAusCitiesThisQuarter` refers to the measure of cells that were grouped by dimension attributes `P` and `T`. This measure represents the average `SalesTotal` of Australian cities of the current quarter.

4.2 Semantics of Analysis Rules

In decision making, an analysis rule examines for each entity e of the analysis scope, whether the rule's specified OLTP method should be invoked in operational systems or whether e should be forwarded to further manual analysis.

To come to a decision, the rule's decision steps are evaluated for e. Since there may be the situation that an analyst did not specify decision steps for every cube of the analysis graph, we introduce for each cube c of the analysis graph V the two conditions $TA_c(e)$ and $DA_c(e)$. $TA_c(e)$ represents the decision to execute the rule's action for e after analyzing c and $DA_c(e)$ represents the decision to continue analysis at more detailed cubes for e after analyzing c. $TA_c(e)$ and $DA_c(e)$ are defined as follows:

$$TA_c(e) = \begin{cases} triggerActionCond_c & : \texttt{if } triggerAction_c \texttt{ is defined for } c \\ \texttt{FALSE} & : \texttt{else} \end{cases}$$

$$DA_c(e) = \begin{cases} detailAnalysisCond_c & : \texttt{if } detailAnalysisCond_c \texttt{ is defined for } c \\ \texttt{TRUE} & : \texttt{else} \end{cases}$$

To mimic the top-down approach in analyzing cubes, we extend the above definitions and introduce $DA_c^+(e)$ as the extended detail analysis condition of cube c and $TA_c^+(e)$ as the extended trigger action condition of cube c. $DA_c^+(e)$ is defined recursively and is satisfied if one of the direct more general cubes of c decides to continue analysis at c and if the detail analysis condition of c is satisfied. $TA_c^+(e)$ is satisfied if the trigger action condition of c is satisfied and at least one direct more general cube of c decides to continue analysis at c.

$$DA_c^+(e) = DA_c(e) \wedge \left(\bigvee_{c_j \in super(c)} DA_{c_j}^+(e) \right)$$

$$TA_c^+(e) = TA_c(e) \wedge \left(\bigvee_{c_j \in super(c)} DA_{c_j}^+(e) \right)$$

The semantics of an analysis rule is thus defined declaratively by the two boolean functions $executeOLTPMethod(e)$ and $analyzeManually(e)$, which determine for an entity e, whether the analysis rule's OLTP method should be executed or whether e has to be analyzed manually. Function $executeOLTPMethod(e)$ evaluates to "TRUE" if analyzing the cells of at least one cube of the analysis graph results in the decision to trigger the rule's action for e. Function $analyzeManually(e)$ evaluates to "TRUE" if analyzing the cells of at least one cube of the analysis graph results in the decision to detail analysis *and* no other cube decides to trigger the rule's action for e.

1. $executeOLTPMethod(e) = \bigvee_{c \in V} TA_c^+(e)$
2. $analyzeManually(e) = \left(\bigvee_{c \in V} DA_c^+(e) \right) \wedge (executeOLTPMethod(e) =$ "FALSE")

5 Conclusion and Future Work

In this paper, we introduced a novel architecture for automatic analysis and decision making in data warehouses. We called this architecture *active data warehouse*. We further introduced analysis rules, which are templates for automatic decision making in active data warehouses. Analysis rules fire at the occurrence of a calendar event or at the occurrence of a relative temporal event, which may refer to method invocations in operational systems (i.e., OLTP method events). Automatic decision making of analysis rules is accomplished by using (1) the analysis graph, which is a hierarchy of cubes and (2) by specifying a set of decision steps, which apply decision criteria on these cubes. An analysis rule's action

will be executed in an OLTP system for an entity if the rule considers the entity to be "decided". We have built a proof-of-concept prototype on top of an existing relational database system (Oracle 8i), which will be described in a forthcoming paper.

References

1. R. Agrawal, A. Gupta, and S. Sarawagi. Modeling Multidimensional Databases. In A. Gray and P. Larson, editors, *Proc. of the 13th Intl. Conf. on Data Engineering, April 7-11, 1997 Birmingham U.K*, pages 232–243. IEEE Computer Society Press.
2. L. Cabibbo and R. Torlone. A Logical Approach to Multidimensional Databases. In *EDBT'98, Proc. of 6th Intl. Conf. on Extending Database Technology, March 23-27, 1998, Valencia, Spain*, pages 183–197. Sringer LNCS 1377.
3. S. Castangia, G. Guerrini, D. Montesi, and G. Rodriguez. Design and Implementation for the Active Rule Language of Chimera. In N. Revell and A M. Tjoa, editors, *Proc. of the Workshop on Database and Expert Systems Applications (DEXA)*. OMNIPRESS, 1995.
4. S. Chakravarthy and D. Mishra. Snoop: An Expressive Event Specification Language for Active Databases. *Data & Knowledge Engineering*, 14(1):1–26, November 1994.
5. S. Chaudhuri and U. Dayal. An Overview of Data Warehousing and OLAP Technology. *ACM SIGMOD Record*, 26(1):65–74, March 1997.
6. E.F. Codd, S.B. Codd, and C.T. Sally. *Providing OLAP (On-Line Analytical Processing) To User Analysts: An IT Mandate*. Arbor Software Corporation, White Paper, 1993.
7. M. Golfarelli, D. Maio, and S. Rizzi. Conceptual Design of Data Warehouses from E/R Schemes. In H. El-Rewini, editor, *HICSS'98, Proc. of the 31st Hawaii Intl. Conf. on System Sciences, Volume VII, 1998*, pages 334–343. IEEE Computer Society Press.
8. J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, and M. Venkatrao. Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals. *Data Mining and Knowledge Discovery*, 1(1):29 – 53, March 1997.
9. W. Lehner. Modeling Large Scale OLAP Scenarios. In *EDBT'98, Proc. of 6th Intl. Conf. on Extending Database Technology, March 23-27, 1998, Valencia, Spain*, pages 153–167. Springer LNCS 1377.
10. N. Paton and O. Diaz. Active database systems. *ACM Computing Surveys*, 31(1):63 – 103, March 1999.
11. P. Vassiliadis. Modeling Multidimensional Databases, Cubes and Cube Operations. In M. Rafanelli and M. Jarke, editors, *10th Intl. Conf. on Scientific and Statistical Database Management, Proceedings, Capri, Italy, July 1-3, 1998*, pages 53–62. IEEE Computer Society Press.
12. P. Vassiliadis and T. Sellis. A Survey of Logical Models for OLAP Databases. *ACM SIGMOD Record*, 28(4):64–69, December 1999.

Supporting Hot Spots with Materialized Views

J. Albrecht, A. Bauer, M. Redert
Department of Database Systems, University of Erlangen-Nuremberg
Martensstr. 3, 91058 Erlangen, Germany
{albrecht, andreas, mlredert}@immd6.informatik.uni-erlangen.de

Abstract. Since data warehousing has become a major field of research there has been a lot of interest in the selection of materialized views for query optimization. The problem is to find the set of materialized views which yields the highest cost savings for a given set of queries under a certain space constraint. The analytical perspective results in queries which on the one hand require aggregations but on the other hand are quite restrictive with regard to the fact data. Usually there are "hot spots", i.e. regions which are requested very frequently, like the current period or the most important product group. However, most algorithms in literature do not consider restrictions of queries and therefore generate only views containing all summary data at a certain aggregation level although the space it occupies could better be used for other, more beneficial views. This article presents an algorithm for the selection of restricted views. The cost savings using this algorithm have been experimentally evaluated to be up to 80% by supplying only 5% additional space.

1 Introduction

A very popular technique to increase the performance of data warehouse queries is the precomputation of results in the form of materialized views. To gain maximum performance for a given set of queries the "best" set of views which fits into a supplied amount of space has to be determined. For a given query behaviour the view selection problem can be stated as follows:

Definition 1: Let $Q = \{Q_1, ..., Q_n\}$ be a set of queries with the respective access frequencies $\{f_1, ..., f_n\}$ and let S be a supplied amount of space for materialized views. A solution to the *view selection problem* is a set of views $\mathcal{V} = \{V_1, ..., V_m\}$ with $\sum_i |V_i| \leq S$ such that the total cost to answer all queries $C(Q) = \sum_i f_i C(Q_i)$ is minimal.[1]

Several preaggregation mechanisms exist for the view selection problem (see section 5). Most of them are based on an aggregation lattice ([HaRU96]). Informally an aggregation lattice is a DAG showing the dependencies among the sets of grouping attributes (granularities) in a multidimensional database. In the aggregation lattice in figure 1 the higher (coarser) nodes can be computed from the lower (finer) nodes. A more formal treatment of the relationship between the nodes is given in the next section.

[1] The partitioning algorithm which is presented in this article aims at minimizing query cost. An extension to include update costs is straightforward as illustrated for example in [BaPT97].

Y. Kambayashi, M. Mohania, and A M. Tjoa (Eds.): DaWaK 2000, LNCS 1874, pp. 47–56, 2000.
© Springer-Verlag Berlin Heidelberg 2000

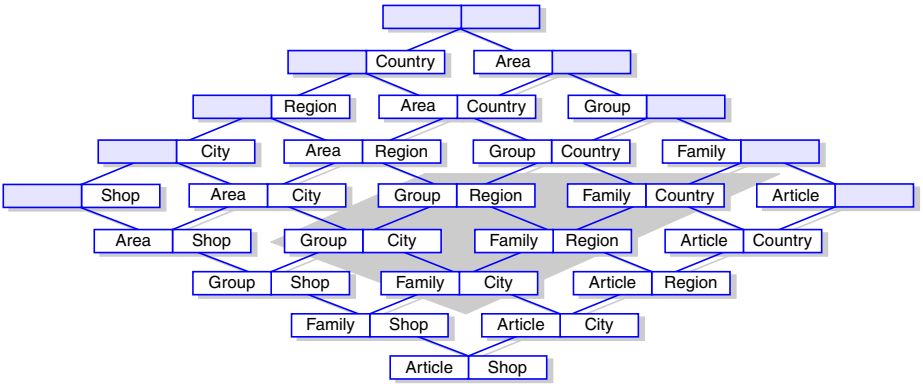


Fig. 1. An aggregation lattice: The direction of the edges is from lower to higher nodes.

To illustrate the drawbacks of lattice-based algorithms as well as to simplify our examples we will restrict the discussion to multidimensional databases in the sense of [BaPT97] represented by a star/snowflake schema ([Kimb96]). The general structure of an aggregate query on a star schema (star query) is

```
SELECT    <grouping attributes>, AGG(<measure>)
FROM      <fact table>, <dimension tables>
WHERE     <join conditions> AND
          <restrictions>
GROUP BY  <grouping attributes>
```

We call queries which have restrictions *restricted*. If there are only join conditions in the where clause, the queries are called *unrestricted*. Lattice-based algorithms like [HaRU96] and [BaPT97] treat all queries as unrestricted queries, restrictions are just not considered. The queries are mapped to the nodes in the lattice which only reflect the grouping attributes. Based on the reference frequencies of each node it is decided which of the (unrestricted) nodes are to be materialized. Dealing only with grouping attributes allows a relatively simple view selection algorithm.

But most of the data warehouse queries are focused on certain product groups, locations, customer segments and especially time ranges. And there are hot spots with respect to these restrictions, too. Usually the current time period is most interesting. If for example 80% of the queries concern only the last month, why should one materialize an aggregate containing data for the last 5 years as well? Besides these obvious time restrictions there are very often “hot” product groups, customer segments and so on. Neglecting these hot spots results in two major drawbacks: First, space is wasted on one aggregation level which could be used for more interesting regions on other aggregation level, and second, the unrestricted materialized views are much larger as restricted views. Therefore query runtime increases.

If the query behaviour is known or can be estimated, a view selection algorithm should produce the “best approximation” of the queries under a given space constraint. In the next sections we present an algorithm which computes a set of restricted aggregates which is “tailored” for a certain set of queries. The basic idea is to select only useful

partitions or fragments of each lattice node. Restrictions in data warehouse queries, like "video devices in Germany" or "Consumer Electronis in December 1999", are based on the dimension hierarchies. Therefore, the split algorithm is also designed to partition the data space along these classification boundaries.

The rest of the paper is organized as follows: The formal framework needed for our method is introduced in the next section. The actual algorithm follows in section 3. Section 4 shows some experimental results. Some related work is mentioned briefly in section 5. The paper concludes with a summary.

2 Dimensions, Granularities and Queries

This section introduces some notation which is necessary for the simple description of queries and their relationships.

Dimensions. A fundamental aspect of a multidimensional database is that the items in a dimension are organized hierarchically. The structure of a hierarchy is defined by dimensional schemas with dimensional attributes like product groups, cities etc.

Definition: Let "$\rightarrow$" denote the functional dependency. A *dimension schema* is a partially ordered set of attributes, called *dimension levels,* $\mathcal{D}=[\{D_1,...,D_n\}, \rightarrow]$ with a least element[2].

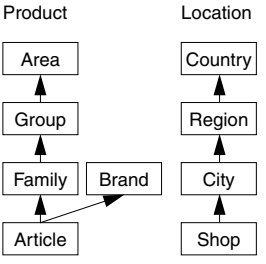


Fig. 2. Dimension schemas.

Figure 2 shows two sample dimensional schemas for products and locations. Parallel hierarchies like the brand classification on the products are also possible. To distinguish the attributes from different dimensions we will use the notation $\mathcal{D}.D_j$, e.g. Product.Brand or simply P.Brand.

Granularities. The partial orders on the dimensional schemas induce a partial order on the power set of all attributes from all dimensions. The set of grouping attributes of a star query, called *granularity* or *aggregation level*, is an element of this power set. More formal, a granularity is a set of attributes $G=\{g_1, ..., g_n\}$ where $g_i=\mathcal{D}.D_j$ in some dimension $\mathcal{D}$ with a partial order based on on the functional dependencies.

Definition: A granularity $G=\{g_1,...,g_n\}$ is *finer than or equal to* a granularity $G'=\{g_1',...,g_k'\}$, denoted as $G \leq G'$, if and only if for each $g_j' \in G'$ there is a $g_i \in G$ such that $G_i \rightarrow G_j'$.

This partial order is the basis for the aggregation lattice ([HaRU96], [BaPT97]). For example {P.Article, L.Shop} $\leq$ {P.Family, L.City} $\leq$ {L.Region} (figures 1 and 2). The partial order can also be used to define a greatest common ancestor of two granularities which is unique because of the least element in each dimension schema.

---

[2] The functional dependency only induces a partial order, if there are no attributes $A \rightarrow B$ and $B \rightarrow A$ with $A \neq B$, which is assumed in the following.

Definition: The *greatest common ancestor* of two granularities G_1 and G_2, denoted as $G_1 \oplus G_2$ is the granularity G such that

- $G \leq G_1$ and $G \leq G_2$ and
- there is no G' with $G' \leq G_1$ and $G' \leq G_2$ and $G \leq G'$.

Queries. If a single measure with an additive aggregation function ([AlGL99]) and lossless joins are assumed, star queries can simply be characterized by the granularity and the restrictions denoting the scope of interest.

Definition: A query Q consists of a granularity G and a scope S, i.e. Q=[G, S].

In our examples we assume the scope to be a conjunction of restrictions of the kind $\mathcal{D}.D_j$ = <value>, for example Product.Group = "Video". Note, that this is not a prerequisite of the algorithm. If an efficient algorithm is used to check containment for more complex predicates, e.g. [AlGL99], more general restrictions could be used as well.

3 The Selection of Aggregate Partitions

In preparation of the partitition selection algorithm the requested scopes of the queries must be reflected in the aggregation lattice. Therefore, the node definition must be extended.

3.1 Building the Lattice

Because lattice based algorithms face a scalability problem ([BaPT97]), we will not introduce more nodes, but just attach the requested scopes and their respective reference frequencies to the node. Thus a node can still be uniquely identified by its granularity.

Definition: A *lattice node* is defined as a structure $N=[G, \mathcal{S}, \mathcal{R}]$ where G is its granularity, $\mathcal{S} = \{S_1, ..., S_n\}$ is a set of scopes and $\mathcal{R} = \{R_{S1}, ..., R_{Sn}\}$ are the reference frequencies for the scopes.

To build the reduced lattice for a set of reference queries Q, the algorithm described in [BaPT97] can be used with small changes. The algorithm is illustrated for nodes without reference frequencies, these are added in the next step (section 3.2). First for each granularity G_i appearing in a query a node containing all referenced scopes must be created. If there is already such a node, only the scope of the query is added to the scope set of the node. Otherwise the respective node is created. In the second part of the algorithm for any two nodes the greatest common ancestor with the superset of the scopes is added to the lattice.

3.2 Propagation of Reference Frequencies

The algorithm for the partitioned view materialization faces a hen and egg problem. On the one hand, the nodes must first be partitioned in order to run the selection algorithm. On the other hand, the partitioning algorithm must know for the computation of the partitioning scheme which views will be materialized in the upper nodes. Consider the nodes A, B and C in figure 3a. B and C represent two queries requesting the scopes S_B

and S_C respectively. Node A=[$G_B \oplus G_C$, {S_B, S_C}] is the the greatest common ancestor of B and C. Because B and C are the only queries, useful partitionings of A might be {S_B, $S_C \backslash S_B$}, {$S_B \backslash S_C$, S_C} or {$S_B \backslash S_C$, $S_C \backslash S_B$, $S_C \cap S_B$}.

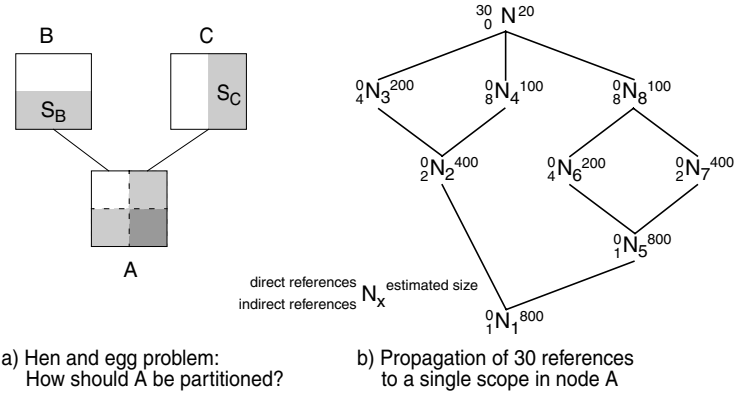


Fig. 3. Propagation of reference freqencies

However, if for example B was materialized, A will never be used to compute B. Thus the reference frequency of $S_B \backslash S_C$ in this case will be zero and therefore only the materialization of the partition for S_C would make sense. But the reference frequencies must be determined *before* the view selection algorithm can be run, because the goal is to materialize those partitions which are referenced most frequently!

Thus we use a heuristic propagation algorithm which distributes the actual access frequencies of a node (*direct references*) to its ancestors (*indirect references*) to estimate the possible references after the view selection. To illustrate this idea consider figure 3b, where a certain scope S was referenced 30 times in node N. These thirty references have to be propagated to the other nodes, which in the example have zero direct references for S. The underlying assumption for the propagation algorithm is that the smaller the cardinality (expected number of tuples) of an ancestor node, the more likely this node will be materialized. Let N_1, ..., N_n be the ancestors of the node N with the direct references $R_{N,S}$ to scope S. Then the number of indirect references $\hat{R}_{N_k,S}$ is computed by $\hat{R}_{N_k,S} = R_{N,S} \cdot \frac{1}{|N_k|} \Big/ \sum_{i=1}^{n} \frac{1}{|N_i|}$. For example we get $\hat{R}_{N_2,S} = 30 \cdot \frac{1}{400} \Big/ \frac{30}{800} = 1$ for node N_2 in the lattice. This number is added to the number of direct references of S in each ancestor N_K yielding the total number of references which is used for the partitioning algorithm.

3.3 Partitioning the Lattice Nodes

In this step each node of the reduced lattice will be recursively split up into partitions. The optimal size of the partitions depends on the reference frequencies of the scopes in this node. However, accessing many partitions does imply some overhead for their unification and this is why it makes sense to define a cost function. This offers the ability to compare the cost of accessing several fragments with the access cost of a single table. The idea of the split algorithm is then to split the fact table in each node as long as the overall access cost of all queries to the fragments gets cheaper.

Cost function

Assume that the node N=[G, {S_1, ..., S_n}, {R_{S1}, ..., R_{Sn}}] was already partitioned into the scope partitions $\mathcal{P}$ = {P_1, ...,P_q}[3]. Then the cost C(S_i, $\mathcal{P}$) to read scope S_i from P_1, ...,P_q consists of two parts: scanning costs and unification costs.

Actually, the formulas for these costs do not matter as long as they are monotonous. In correspondence to [HaRU96] we assumed that the scanning cost for reading a partition grows linearly with its size. The unification cost grows linearly with the intersection of S and the respective partitions because these tuples have to be written to disk temporarily. Assuming that S overlaps partitions P_j, ..., P_k we used the following estimate for the access cost: $C(S, \mathcal{P}) = \sum_{i=j}^{k} (a_s \cdot |P_i| + b_s) + (a_u \cdot |P_i \cap Q| + b_u)$

The constants a_s and a_u (scan and update) for the time to scan one tuple and b_s and b_u for the overhead to open and close the table are system dependent.

The total cost C of accessing all scopes {S_1, ..., S_n} in node N is the sum of the costs for each scope weighted by the reference frequencies: $C(N, \mathcal{P}) = \sum_{i=1}^{n} R_{S_i} \cdot C(S_i, \mathcal{P})$

Partitioning Algorithm

Let N=[G, {S_1, ..., S_n}, {R_{S1}, ..., R_{Sn}}] be the node to be partitioned. The recursive partitioning algorithm is shown in figure 4. It is initialized with the call

Partition(N, ∅, TopLevels(G)),

i.e. it starts with a single partition representing the whole node without any restrictions. This partition is then recursively split into smaller pieces by using the semantic grid structure given by the classification hierarchies. The split process starts at the highest dimension levels. This is expressed by TopLevels(G). For example (figure 2)

Algorithm: Partition (N, P, $\mathcal{L}$ **)**

Input: node N=[G, {S_1, ..., S_n}, {R_1, ..., R_n}]
the partition P to be split
set of hierarchy levels $\mathcal{L}$={L_1, ..., L_l}
Output: partitioning $\mathcal{P}$

```
1  Begin
2    Cmin = ∞
3
4    // try iteratively to split in each dimension
5    Foreach level Li∈ L
6      P := split(P, Li)
7
8      // determine partitioning with least cost
9      If ( C(N, P) < Cmin ) Then
10       Cmin := C(N, P)
11       Pmin := P
12       p := i
13     End If
14   End Foreach
15   // if partitioning is cheaper
16   If (Cmin < ∞) Then
17     // adjust split levels
18     L := L \ {Lp} ∪ {Lp-1}
19
20     // run partitioning algorithm recursively
21     P := ∅
22     Foreach P∈ Pmin
23       P := P ∪ Partition( N, P, L )
24     End Foreach
25   End If
26
27   Return P
28 End
```

Fig. 4. Partitioning algorithm.

[3] $\mathcal{P}$ is a partitioning of the whole data space, i.e. $P_i \cap P_j = \emptyset$ and $\cup_i P_i$ yields the whole space.

TopLevels({P.Article, S.Shop}) = {P.Area, P.Brand, S.Country} and
TopLevels({P.Family}) = {P.Area}.

To explain the algorithm we will use the example in figure 5 with the node at granularity {P.Article, S.City}, starting with $\mathcal{L}$={P.Area, S.Country} as split levels (brand is omitted for simplicity). The illustrated node has possible requests to the scopes S_1, S_2, S_3 and S_4, which have conjunctive restrictions based on the classification hierarchies, e.g. S_4 = (P.Family = "HomeVCR" AND S.Region="North").

In the first two steps the node is split at P.Area and the left partition again at S.Country to separate those partitions which are never requested (striped). We use the remaining part, i.e. the partition P = {(P.Area="Consumer Electronics" AND S.Country="Germany")} to explain the algorithm. The split level vector is now $\mathcal{L}$ = {P.Group, S.Region}. Thus, the costs C(N, $\mathcal{P}'$) for the following two partitionings $\mathcal{P}$ are computed (line 6):

- $\mathcal{P}_1$ = {("Audio" AND "Germany"), ("Video" AND "Germany")}
- $\mathcal{P}_2$ = {("Consumer Electronics" AND "North"), ("Consumer Electronics AND "South")}

Using $\mathcal{P}_1$ scopes S_1 and S_3 require a scan of both partitions. With $\mathcal{P}_2$ this is only the case for S_1. Here S_2, S_3 and S_4 can each be computed from a single partition which is, according to the partitioning cost computation, cheaper. Therefore the second scheme is applied. After the set of the split levels is adjusted to $\mathcal{L}$ = {P.Group, S.City} (line 18). The algorithm now recursively continues for each partition in $\mathcal{P}_2$. It turns out that it is not useful to split P_1 again. P_2 however is split again into P_3 and P_4 (figure 5).

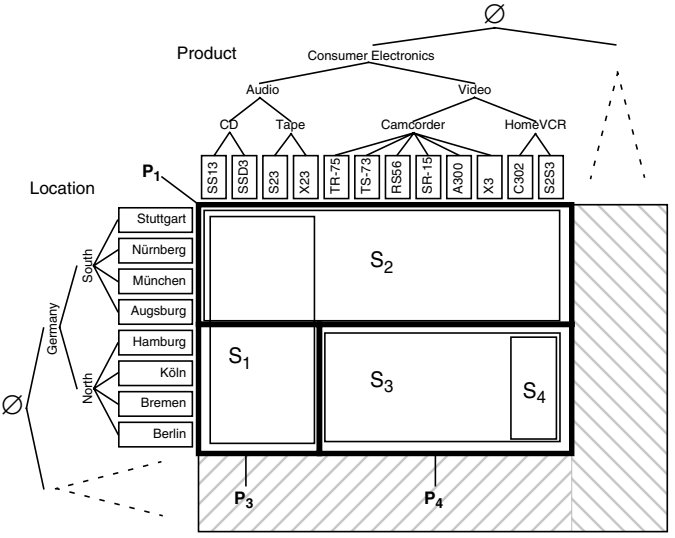
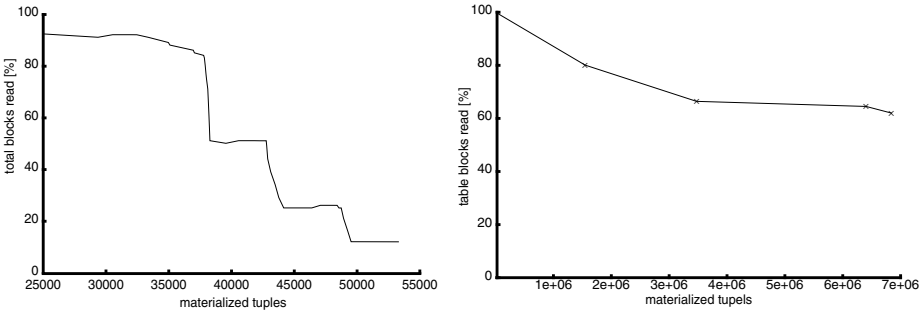


Fig. 5. Illustration of partioning algorithm.

3.4 The Selection of Aggregate Partitions

A preliminary adjustment is necessary before the actual greedy algorithm can work. Consider again scope S_1 in figure 5. If only P_1 was materialized the raw data have to be accessed for the part covered by P_3. If a scan of the raw data is necessary anyway, then S_1 should better be retrieved completely from the raw data. But in this case P_1 would not contribute anything for the computation of S_1 and the benefit is zero. So P_1 only



a) read blocks;100 unrelated queries, partitioned

b) read blocks;100 unrelated queries, unpartitioned

Fig. 6. Experimental results.

yields a benefit for S_1 if also P_3 was materialized! To reflect this, for each scope (each query) in every node a *bucket* containing all partitions which are needed to compute this scope is defined. The bucket for S_1 is for example $\{P_1, P_3\}$.

For the selection of view we use now the greedy algorithm presented in [HaRU96]. In our case however, the objects to materialize are not unrestricted queries, but partitions in a bucket. The benefit is computed on buckets, but if a bucket is selected each partition is materialized separately. The cost model is very similar to the one defined in [HaRU96]. The benefit of a bucket is calculated as the difference of tuples to be scanned to answer all queries with and without the partitions in the bucket materialized.

The algorithm chooses the unmaterialized bucket with the highest benefit based on the already selected buckets. This step is repeated until the available space for preaggregates is exhausted.

4 Experimental Results

Because the algorithm defined in the previous section works on aggregate partitions, the number of blocks to be read in order to answer a given set of queries should drastically decrease. In this part we will demonstrate the efficiency of our method and compare it to the results of the algorithm of [BaPT97].

We used the scenario of the APB-1 benchmark of the OLAP-Council [OLAP98] in a slightly modified version. The model has four dimensions and five measures. The fact table contained about 2.7 million tuples at a sparsity of 0.1%. The measurements were performed on Oracle 8i on a Sun 4000. The system parameters a and b for the partitioning cost function (section 3.3) were determined by several tests in advance.

In the test the input to the algorithms were 100 queries independently requesting arbitrary data sets. Therefore, we do not have a hot spot, but several hot regions. As illustrated in figure 7a, the number of blocks read drops by more than 80%. However in this case approximately 50.000 tuples (still only about 2% of the fact table!) were supplied for materialized aggregates. The borders of the buckets can be clearly identified by the steps in the figure. If not all partitions of a bucket are materialized no further benefit can

be achieved. If it is completely materialized the number of blocks and the time needed to answer the queries drops instantly.

To compare our algorithm with the unpartitioned lattice-based method of [BaPT97], we used the same set of queries and disabled the partitioning. The result is shown in figure 6b. As can be seen about seven million tuples must be materialized to reduce the number of blocks read by 40%.

We did not use indexes or partitioning on the fact table in the experiments. Otherwise the performance gain would not be so dramatic. But still the results would be much better as with former lattice based algorithms.

5 Related Work

In the last few years there have been several influential articles on the view selection problem. Of fundamental importance for this article was the work on aggregation lattices of [HaRU96] and [BaPT97]. [GHRU97] additionally considers existing index structures. [Gupt97] generalizes the aggregation lattice in the form of AND/OR-graphs by considering alternative local query execution plans. An efficient simplification of [HaRU96] is the method of [ShDN98]. The method also proposes a partition scheme for the lattice nodes. However, all partitions of the selected nodes in the lattice are completely materialized. Moreover, the partitioning scheme is physical (in terms of disk pages) and not logical which is more useful in the presence of restriction predicates.

Similar to our algorithm is the one presented in [EzBa98]. Here the predicates in the restrictions of queries are resolved into minterms. However, the number of minterms grows exponentially with the number of queries. A drawback of the algorithm is that the minterms have no relationship to the dimension hierarchies. Therefore, the partitioning scheme will have a relatively arbitrary structure which might even be disadvantegeous for queries not in the reference set. The algorithm of [YaKL97] works on relational algebra expressions. It computes a multple view processing plan for a set of queries and selects subqueries with the highest number of references for precomputation.

The selection of views is not the only problem. Once views are materialized it has to be determined which views are the best to be used to answer a certain query. Coping with the view usage problem are for example the methods presented in [SDJL96], [CoNS99], [AlGL99].

6 Summary

The article presents a new algorithm for the selection of materialized aggregates. By taking into account the restrictions of the queries to be optimized it is possible to come up with a much better approximation of the queries as in previous approaches. Therefore, the algorithm computes with the same space limit more, but smaller aggregates. No space is wasted, regions which are never queried are not materialized. The performance gains are up to 90% in terms of the number of disk blocks.

The partitioning algorithm can also be used to partition the raw data in the fact table without the goal to materialize views, because what the algorithm actually does is find the optimal partitioning of an arbitrary table for a given set of queries. This table does not necessarily have to represent an aggregated view.

References

ABD+99 Albrecht, J.; Bauer, A.; Deyerling, O.; Günzel, H.; Hümmer, W.; Lehner, W.; Schlesinger, L.: Management of multidimensional Aggregates for efficient Online Analytical Processing, in: *International Database Engineering and Applications Symposium* (IDEAS'99, Montreal, Canada, August 1-3), 1999

AlGL99 Albrecht, J.; Günzel, H.; Lehner, W.: Set-Derivability of Multidimensional Aggregates, in: *Proceedings of the First International Conference on Data Warehousing and Knowledge Discovery* (DaWaK'99, Florence, Italy, August 30 - September 1), 1999

BaPT97 Baralis, E.; Paraboschi, S.; Teniente, E.: Materialized View Selection in a Multidimensional Database, in: *Proceedings of the 23rd International Conference on Very Large Data Bases* (VLDB '97, Athens, Greece, August 25-27), 1997

CoNS99 Cohen, S.; Nutt, W.; Serebrenik, A.: Rewriting Aggregate Queries Using Views, in: *Proceedings of the Eighteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems* (PODS '99, Philadelphia (PA), USA, May 31 - June 2), 1999

EzBa98 Ezeife, C.I.; Baksh, S.: A Partition-Selection Scheme for Warehouse Views, in: *Proceedings of the 9th International Conference on Computing and Information* (ICCI '98, Winnipeg, Canada, June 17-20), 1998

GHRU97 Gupta, H.; Harinarayan, V.; Rajaraman, A.; Ullman, J.D.: Index Selection for OLAP, in: *Proceedings of the 13th International Conference on Data Engineering* (ICDE '97, Birmingham, U.K., April 7-11), 1997

Gupt97 Gupta, H.: Selection of Views to Materialize in a Data Warehouse, in: *Proceedings of the 6th International Conference on Database Theory* (ICDT '97, Delphi, Greece, January 8-10), 1997

HaRU96 Harinarayan, V.; Rajaraman, A.; Ullman, J.D.: Implementing Data Cubes Efficiently, in: *Proceedings of the 25th International Conference on Management of Data*, (SIGMOD '96, Montreal, Canada, June 4-6), 1996

Kimb96 Kimball, R.: *The Data Warehouse Toolkit*, 2nd Edition, John Wiley & Sons, Inc., New York, 1996

SDJL96 Srivastava, D.; Dar, S.; Jagadish, H.V.; Levy, A.Y.: Answering Queries with Aggregation Using Views, in: *Proceedings of 22th International Conference on Very Large Data Bases* (VLDB'96, Bombay, India, September 3-6), 1996

ShDN98 Shukla, A.; Deshpande, P.; Naughton, J.: Materialized View Selection for Multidimensional Datasets, in: *Proceedings of the 24th International Conference on Very Large Databases* (VLDB '98, New York (NY), USA, August 24-27), 1998

YaKL97 Yang, J.; Karlapalem, K.; Li, Q.: Algorithms for Materialized View Design in Data Warehousing Environment, in: *Proceedings of the 23rd International Conference on Very Large Data Bases* (VLDB '97, Athens, Greece, August 25-29), 1997

Evaluation of Materialized View Indexing in Data Warehousing Environments

Ladjel Bellatreche[1] and Kamalakar Karlapalem[1] and Qing Li[2]

[1] University of Science and Technology, Clear Water Bay Kowloon, Hong Kong
E-mail : {ladjel, kamal}@cs.ust.hk
[2] City University of Hong Kong, Tat Chee Avenue, Kowloon, Hong Kong
E-mail: csqli@cityu.edu.hk

Abstract. Index selection is one of the most important decisions in designing a data warehouse (DW). In this paper, we present a framework for a class of graph join indices used for indexing queries defined on materialized views. We develop storage cost needed for these indices, and query processing strategies using them. We formulate the graph join index selection problem, and present algorithms which can provide good query performance under limited storage space for the indices. We also evaluate these algorithms to show their utilities by using an example taken from Informix white paper.

1 Introduction

In recent years, there has been an explosive growth in the use of databases for decision support systems which aims to provide answers to complex queries posed over very large databases. Data warehousing is a "collection of support technologies aimed at enabling the knowledge worker to make better and faster decisions" [2]. DWs are designed for decision support purposes and certain large amounts of historical data. For this reason, DWs tend to be extremely large [2]. The OLAP queries (joins and aggregations) are very expensive in computation and they can take many hours or even days to run [6]. This is because queries are performed on tables having potentially millions of records. The join operation of the relational database model is the fundamental operation that allows information for different relations to be combined. Joins are typically expensive operations, particularly when the relations involved are substantially larger than main memory [8, 15]. The star join index (SJI) [13] has been used in DWs modeled by a star schema. It denotes a join index between a *fact table* and multiple *dimension tables*. The SJI has been proved to be an efficient access structure for speeding up joins defined on fact and dimension tables in DWs [13, 9].

Another technique that can be used to speed up the query processing is *materialized views* [5, 14, 16]. The problem of selecting a set of materialized views for a given set of OLAP queries is called *view selection problem*. It starts with a DW schema with dimension tables and fact table(s), and then selects a set of materialized views to speed up a set of OLAP queries under some resource constraint (like disk space, computation time, or maintenance cost).

Y. Kambayashi, M. Mohania, and A M. Tjoa (Eds.): DaWaK 2000, LNCS 1874, pp. 57–66, 2000.
© Springer-Verlag Berlin Heidelberg 2000

Once materialized views are selected, all queries will be rewritten using these materialized views (this process is known as *query rewriting* [12]). A rewriting of a query Q using views is a query expression Q' that refers to these views. In SQL, after rewriting process of OLAP queries, we can find in the FROM clause a set of materialized views, dimension tables and the fact table. These views can be joined each others or with other tables.

The problem of selecting a set of indices to speed up a given set of queries is know as *index selection problem*. It starts with a DW schema, and then builds indexes on top of this schema to speed up a set of OLAP queries under some resource constraint. Among the selected indices, we can find indices defined on the top of a single table and view, and join indices defined on a fact table and dimension tables (e.g., SJI). Indexing a single materialized view can be done by using the same indexing techniques for a table (e.g., B^+-tree). However, there has not been much research reported in enhancing SJI algorithms for efficiently selecting and performing join indices with materialized views. In this paper, we consider the problem of join index selection in the presence of materialized views (PJISV).

1.1 Contributions and Organization of the Paper

To the best of our knowledge, this is the first paper studying the concept of PJISV in DW environments. The main contributions of this paper are that we:

- present a novel indexing technique called *graph join index* to handle the PJISV. This index can be constructed using dimension tables, materialized views and a fact table. This has been discussed in Section 2.
- formulate the PJISV and present algorithms for selecting an optimal or near optimal indices for a given set of OLAP queries. This has been discussed in Section 3.
- show the utility of PJISV through evaluation. This has been discussed in Section 4.

1.2 Related Work

The problem of indexing materialized views was first studied by Roussopoulos [10]. A view index is similar to a materialized view except that instead of storing tuples in the view directly, each tuple in the view index consists of pointers to the tuples in the base relations that derive the view tuple. But the author did not consider the problem of join indexing materialized views. Choosing indices for materialized views is a straightforward extension of the techniques used for indexing tables.

Segev et al. [11] presented a join index called join pattern index in rule environment where changes in the underlying data are propagated *immediately* to the view. This index exists an *all* attributes involved in selection and join conditions. It refers to an array of variables that represent the key and join attributes of the data associated with the join. However, in DWs, a large number of changes are

propagated at once, and the cost of maintaining the indexes often out-weights any benefit obtained by doing join indices. So it is not correct to assume that indices exist on all attributes involved in selection and join conditions [7].

1.3 Motivating Example

We present an example of a DW schema models the activities of a world-wide wholesales company, and will be used as a running example through this paper. This schema consists of three dimension tables *CUSTOMER*, *PRODUCT*, and *TIME*, and one fact table *SALES*. The tables and attributes of the schema are shown in Figure 1.

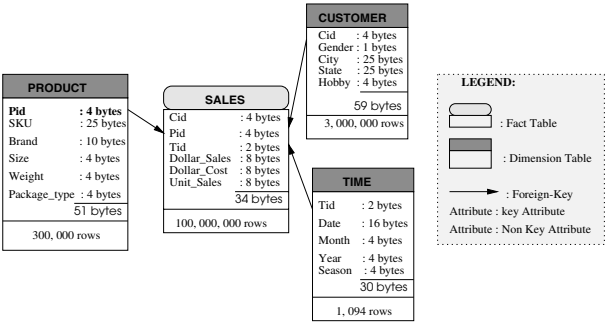


Fig. 1. An Example of a Star Schema

Assume that we are interested in determining the total sales for male customers purchasing product of type package "box". The following SQL query Q_1 may be used for this purpose:

```
SELECT   SUM(S.dollar_sales)
FROM     CUSTOMER C, PRODUCT P, SALES S,
WHERE    C.Cid = S.Cid
AND      P.Pid = S.Pid
AND      C.Gender ='M'
AND      P.Package_type = ''Box''
GROUP BY PID
```

This company also maintains the following materialized view that consists in finding the total sales for each product having "box" as type of package. This view is defined as follows:

```
CREATE VIEW V_2
SELECT   *
FROM     PRODUCT P, SALES S,
WHERE    P.Pid   = S.Pid
AND      C.Package_Type =''Box''
GROUP BY PID
```

The view V_2 can be used to evaluate the query Q_1 by joining V_2 with the dimension table CUSTOMER. The rewritten query Q'_1 that uses V_2 is:

```
SELECT   SUM(S.dollar_sales)
FROM     CUSTOMER C, V_2,
WHERE    V_2.Cid = C.Cid
AND      C.Gender ='M'
GROUP BY PID
```

Note that the fact table SALES is very huge, and the materialized view V_2 is likely to be orders of magnitude smaller than SALES table. Hence, evaluating Q'_1 will be much faster than evaluating the query Q_1. This is because, Q_1 needs two join operations, but Q'_1 needs only one. There is a solution to reduce the number of join operations of the query Q_1 which is the SJI [13]. The suitable SJI for Q_1 is (CUSTOMER, SALES, PRODUCT), but it requires a very high storage capacity [4] that can slow down the execution of the query Q_1. A join index [15] between the view V_1 and the dimension table CUSTOMER can speed up the execution of the query Q_1, and its size is much smaller than the SJI (CUSTOMER, SALES, PRODUCT).

2 Notations & Definitions

Definition1. (Database Schema Graph (DBSG)) is a connected graph where each node is a relation and an edge between two nodes R_i and R_j implies a possibility of join between relations R_i and R_j.

Definition2. (Join Graph (JG)) is a database schema graph such that there exists exactly **one** join condition between two relations. In DW environments, a JG is the same as a DBSG.

Claim 1 *A JG in a DW with only dimension tables and a fact table is always connected. In this case a JG is called* **total join graph**.

Claim 2 *In a DW context with dimension tables, a fact table and materialized views, a JG can be disconnected, i.e., it contains more than one subgraphs. In this case it is called* **partitioned join graph**.

The partitioned join graph occurs when: (1) views do not have common join attributes, (2) views have common join attributes, but they can never be joined.

Definition3. (Graph Join Index (GJI)) A GJI is a subset of nodes of a JG.

In case of distributed join graph, a GJI is a subset of nodes of each sub-graph.

A DW schema consists of a star schema (a set of dimension tables and a fact table) with a set of materialized views. Each table T_i being either a dimension table, a view, or a fact table has a key K_{T_i}. A GJI on m tables is a table having m attributes $(K_{T_1}, K_{T_2}, ..., K_{T_m})$, and is denoted as :
$(T_1 \sim T_2 \sim ... \sim T_m)$.

Since the star schema is connected, there is a path connecting the tables $(T_1, T_2, \ldots, T_m)$ by including additional tables, if necessary. The GJI supplies the keys of the relations $(T_1, T_2, \ldots, T_m)$ that form the result of joining all the tables along the path. In case there are multiple paths that connect tables $(T_1, T_2, \ldots, T_m)$, then there can be multiple GJIs, each catering to a different query. A GJI with all tables of a JG is known as a complete GJI. A GJI with a subset of nodes of a JG is referred as partial GJI (see Figure 2).

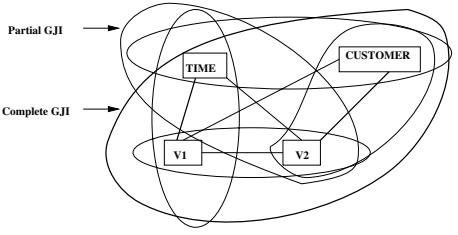


Fig. 2. Join Graph and Graph Join Indices

Note that for a star schema with n tables, the number of possible GJIs grows exponentially, and is given by:

$$\binom{2^n - n - 1}{0} + \binom{2^n - n - 1}{2} + \ldots + \binom{2^n - n - 1}{2^n - n - 1} = 2^{(2^n - n - 1)}$$

3 GJI-Selection (GJI-S) Schemes

In this section, we formulate the GJI-S problem and we present GJI-S algorithms. The GJI-S problem can be viewed as a search problem where the search space consists of all possible indices for a given set of queries. Since the search space is very large ($2^{(2^n - n - 1)}$, for n tables), we need to have algorithms to find the optimal or the "near optimal" GJIs for processing a set of queries.

3.1 GJI-S Problem Formulation

The inputs of our problem are as follows:
(i) A star schema with the fact table F and dimension tables $\{D_1, D_2, ..., D_d\}$, **(ii)** a set of most frequently asked queries $Q = \{Q_1, Q_2, ..., Q_s\}$ with their frequencies $\{f_1, f_2, ..., f_s\}$, **(iii)** a set of materialized views $V = \{V_1, V_2, ..., V_m\}$ selected to support the execution of queries in Q, and **(iv)** a storage constraint S.

The GJI-S problem selects a set of GJIs G', which minimizes the total cost for executing a set of OLAP queries, under the constraint that the total space occupied by GJIs in G' is less than S.

More formally, let $C_g(Q_i)$ denotes the cost of answering a query Q_i using a GJI g ($g \in G'$). The problem is to select a set of GJIs G' ($G' \subseteq G$) [3] that

[3] G represents all possible GJIs for a star schema.

minimizes the total query processing cost ($TQPC$) and under the constraint that: $S_{G'} \leq S$, where: $\mathbf{TQPC} = \sum_{\mathbf{i=1}}^{\mathbf{s}} \mathbf{f_i} \times \{\mathbf{min}_{\forall \mathbf{g} \in \mathbf{G}} \mathbf{C_g}(\mathbf{Q_i})\}$.

3.2 Graph Join Index Selection Algorithms

All index selection algorithms select indices to optimize the objective function employed, subject to the constraint that the indices included cannot consume more than the specified amount of storage. Our objective function is based on two cost models that estimate the number of disk accesses (IOs) to answer a set of queries in the presence of GJIs with and without materialized views. In order to elaborate these cost models, we consider the following assumptions: (1) dimension tables, the fact table, materialized views and indices are stored on the disk (our cost model can easily be adapted when dimension tables, the fact table, materialized views or indices are in the main memory), and (2) selection conditions are always pushed down onto the dimension tables and the fact table as in [7].
Due to lack of space, we omit the description of our cost models. For details, see [1] in which we have also estimated the storage cost needed for the GJI structure.

Naive Algorithm (NA) is used for a comparison purpose. The NA enumerates all possible GJIs, and calculates the contribution of each GJI. Finally, it picks the GJI having the minimum cost while satisfying the storage constraint.

Greedy Algorithm for Selecting one GJI We have developed a heuristic algorithm (GA1) for selecting only one GJI. Given a storage constraint S, our GA1 constructs a weight GJI graph from the GJI graph by adding a weight to each edge which equals to the sum of query frequencies using that edge. Note that each edge represents a join index between its two nodes. The GA1 starts with the simplest join index that corresponds to most frequently executed join among all the queries, and tries to expand it by adding more nodes while checking if the total processing cost decreases. Once no more *expand* operations can be applied, the algorithm tries to *shrink* it to check if a better GJI can be derived. The GA1 keeps repeating these two operations (expand and shrink) till no more reduction in query processing cost can be achieved without violating the storage cost.

Greedy Algorithm for Selecting more than one GJI Note that as a single GJI generated by GA1 can only efficiently execute some but not all of the queries, we need more than one GJI to efficiently execute all the queries. Therefore, we have developed an algorithm GAK that within the storage constraint finds at most $2^n - n - 1$ GJIs which can reduce the query processing cost as much as possible. The GAK algorithm starts with the initial solution provided by GA1 as the first index. After that it selects the edge from weight graph that has the highest query access frequency and which is not a GJI generated by GA1. Then it

tries to expand (one or more times) and shrink (one or more times) till no further reduction in total cost can be achieved and the storage constraint is satisfied. This generates the second GJI, and GAK keeps repeating this procedureuntil at most $(2^n - n - 1)$ indices are found or the storage constraint is violated.

4 Evaluation of Graph Join Indices

Our experimental study utilized a DW schema and data set from [3]. The number of rows and the width (in bytes) of each table are given in Figure 1. The page size (in bytes) that we use for our evaluation is 8192.

We consider five frequently asked queries [1]. To support the execution of these queries, we select by using the Yang et al. algorithm [16] two views V_1 and V_2 to be materialized such as:

- $V_1 = \sigma_{State=\text{“}IL\text{”}}(CUSTOMER) \bowtie$ SALES, and
- $V_2 = \sigma_{Package=\text{“}Box\text{”}}(Product) \bowtie$ SALES.

The matrix in Figure 3 represents the rewriting matrix. It also gives the frequency (FR) with which each query is executed.

	V1	V2	T	C	FR
Q1	1	0	0	0	20
Q2	0	1	1	0	5
Q3	0	1	1	1	5
Q4	1	0	1	0	10
Q5	0	1	0	1	15

Fig. 3. Rewriting Matrix

Since the queries can directly utilize these materialized views, we first evaluate whether indexing the views and dimension tables leads to any reduction of query processing cost. Table 1 enumerates all 11 possible GJIs (for $n = 4$ nodes). The GJIs are built over dimension tables *TIME (T)* and *CUSTOMER (C)*, and views $View_1$ *(V_1)* and $View_2$ *(V_2)*. For example, in Table 1, GJI $(V_1 \sim V_2)$ is between views V_1 and V_2, whereas the GJI $(V_1 \sim T \sim C)$ is between view V_1 and dimension tables *TIME* and *CUSTOMER*, and the GJI $(V_1 \sim V_2 \sim T \sim C)$ is the complete GJI covering views V_1 and V_2, and dimension tables *TIME* and *CUSTOMER*. Note that the cardinality of views V_1 and V_2 are $200,000$ and $400,000$ rows, respectively. Table 1 also lists the number of rows for GJIs.

Referring to Table 2, one can see that the GJI $(V_1 \sim T)$ has the least storage cost 81504731 bytes, but it gives the least query processing cost for only query Q_4. Note that the GJI $(V_1 \sim V_2 \sim T \sim C)$ gives the least query processing cost for queries Q_1, Q_2, Q_3 and Q_5, but not for query Q_4. This is because Q_4 needs to access only V_1 and $TIME$, and the GJI $(V_1 \sim T)$ fits this query exactly. The storage requirement for $(V_1 \sim V_2 \sim T \sim C)$ is of 293778358 bytes (about 0.3 GBs). Note that there is no single GJI which gives the best query processing

Index	View$_1$ (V_1)	View$_2$ (V_2)	TIME (T)	CUSTOMER (C)	# Rows
$V_1 \sim V_2$	1	1	0	0	4000000
$V_1 \sim T$	1	0	1	0	2000000
$V_1 \sim C$	1	0	0	1	2000000
$V_1 \sim V_2 \sim T$	1	1	1	0	4000000
$V_1 \sim V_2 \sim C$	1	1	0	1	4000000
$V_1 \sim T \sim C$	1	0	1	1	2000000
$V_1 \sim V_2 \sim T \sim C$	1	1	1	1	4000000
$V_2 \sim T$	0	1	1	0	4000000
$V_2 \sim C$	0	1	0	1	4000000
$V_2 \sim T \sim C$	0	1	1	1	4000000
$T \sim C$	0	0	1	1	9846000

Table 1. Set of GJIs and their Sizes

cost for all the queries. When indexing materialized views, one can constrain the storage requirement even further. That is, by putting a storage availability of just 0.23 GBs for GJIs, the best possible index ($V_2 \sim T \sim C$) requires just 91821 page accesses as against 510669 page accesses for the hash join algorithm (HJA).

Index	Storage	Q_1	Q_2	Q_3	Q_4	Q_5	TQPC
$V_1 \sim V_2$	197751734	21729	41522	106343	21729	106328	297652
$V_1 \sim T$	81504731	21729	118683	183504	636	151248	475800
$V_1 \sim C$	107653595	21729	118656	141254	21729	119560	422928
$V_1 \sim V_2 \sim T$	232690614	21729	1158	104431	663	106328	234310
$V_1 \sim V_2 \sim C$	258839478	21729	41522	41522	21729	2531	129034
$V_1 \sim T \sim C$	125159899	21729	118683	118683	636	119560	379292
$V_1 \sim V_2 \sim T \sim C$	293778358	21729	1158	1158	663	2531	27240
$V_2 \sim T$	162853814	21729	1158	104431	65244	106328	298891
$V_2 \sim C$	189002678	21729	41522	41522	65190	2531	172495
$V_2 \sim T \sim C$	223941558	21729	1158	1158	65244	2531	91821
$T \sim C$	97060186	21729	118697	118697	65231	119580	443935
Complete SJI	4967170048	23144	31595	31595	23147	45491	154973
Hash Join	0	21729	118656	153846	65190	151248	510669

Table 2. GJI Storage Cost and Query Processing Costs

Table 3 gives the normalized query processing costs for queries Q_1 to Q_5 using different GJIs and a normalized total cost. The normalization is done with respect to HJA for processing the five queries. The complete GJI ($V_1 \sim V_2 \sim T \sim C$) has the cost of processing queries Q_1, Q_2, Q_3, Q_4, and Q_5 ranging from **just 1%** (for query Q_3) to 100% (for query Q_1) of the HJA cost for those queries. Note that the least cost for processing query Q_1 is 100% of the HJA cost. For

queries Q_3 and Q_5, almost all GJIs (except $V_1 \sim T$) provide cost much lesser than the HJA cost (ranging from 1% to 119% for Q_3, and 2% to 100% for Q_5). Therefore, there is a significant cost reduction in processing queries with GJIs in comparison to the HJA approach.

Index	Q_1	Q_2	Q_3	Q_4	Q_5	TQPC
$V_1 \sim V_2$	1.00	0.35	0.69	0.33	0.70	0.58
$V_1 \sim T$	1.00	1.00	1.19	0.01	1.00	0.93
$V_1 \sim C$	1.00	1.00	0.92	0.33	0.79	0.82
$V_1 \sim V_2 \sim T$	1.00	0.01	0.68	0.01	0.70	0.45
$V_1 \sim V_2 \sim C$	1.00	0.35	0.27	0.33	0.02	0.25
$V_1 \sim T \sim C$	1.00	1.00	0.77	0.01	0.79	0.74
$V_1 \sim V_2 \sim T \sim C$	1.00	0.01	0.01	0.01	0.02	0.05
$V_2 \sim T$	1.00	0.01	0.68	1.00	0.70	0.58
$V_2 \sim C$	1.00	0.35	0.27	1.00	0.02	0.33
$V_2 \sim T \sim C$	1.00	0.01	0.01	1.00	0.02	0.17
$T \sim C$	1.00	1.00	0.77	1.00	0.79	0.86

Table 3. Normalized Query Processing Costs

5 Conclusions

Index selection is a very important problem in data warehouse design. In this paper, we have presented the graph join index (GJI) mechanisms for efficiently processing queries in a data warehousing environments. GJIs can be uniformly applied to both fact/dimension tables and materialized views, and facilitate processing of queries both with and without materialized views. We have developed the storage, query processing, and update cost models for GJIs, and furthermore evaluated the GJIs in terms of their storage cost and query processing efficiency. We have also developed, for a given set of queries with limited storage availability, two greedy GJI selection algorithms for both getting only one GJI and more than one GJIs. These algorithms are tested against an example data warehousing environment and are found to give the optimal solution (with respect to the one derived from exhaustive search).

From our results, we find that having GJIs (whether partial or complete) provides better performance than a two-node join index while incurring quite reasonable extra storage overhead.

Future work involves developing efficient query processing strategies using GJIs, and further evaluation and development of index selection algorithms using other techniques, such as index merging and simulated annealing.

References

1. L. Bellatreche, K. Karlapalem, and Q. Li. Evaluation of indexing materialized views in data warehousing environments. *extended paper, available on http://www.cs.ust.hk/~ladjel.*
2. S. Chaudhuri and U. Dayal. An overview of data warehousing and olap technology. *Sigmod Record*, 26(1):65–74, March 1997.
3. Informix Corporation. Informix-online extended parallel server and informix-universal server: A new generation of decision-support indexing for enterprise data warehouses. *White Paper*, 1997.
4. A. Datta, K. Ramamritham, and H. Thomas. Curio: A novel solution for efficient storage and indexing in data warehouses. *Proceedings of the International Conference on Very Large Databases*, pages 730–733, September 1999.
5. A. Gupta and I. S. Mumick. Maintenance of materialized views: Problems, techniques, and applications. *Data Engineering Bulletin*, 18(2):3–18, June 1995.
6. H. Gupta, V. Harinarayan, A. Rajaraman, and J. Ullman. Index selection for olap. *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 208–219, April 1997.
7. W. Labio, D. Quass, and B. Adelberg. Physical database design for data warehouses. *Proceedings of the International Conference on Data Engineering (ICDE)*, 1997.
8. H. Lei and K. A. Ross. Faster joins, self-joins and multi-way joins using join indices. *Data and Knowledge Engineering*, 28(3):277–298, November 1998.
9. P. O'Neil and D. Quass. Improved query performance with variant indexes. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 38–49, May 1997.
10. N. Roussopoulos. View indexing in relational databases. *ACM Transactions on Database Systems*, 7(2):258–290, June 1982.
11. A. Segev and J. L. Zhao. A framework for join pattern indexing in intelligent database systems. *IEEE Transactions on Knowledge and Data Engineering*, 7(6):941–947, December 1995.
12. D Srivastava, S. Dar, H. Jagadish, and A. Y. Levy. Answering queries with aggregation using views. *Proceedings of the International Conference on Very Large Databases*, pages 318–329, 1996.
13. Red Brick Systems. Star schema processing for complex queries. *White Paper*, July 1997.
14. D. Theodoratos and T. K. Sellis. Data warehouse configuration. *Proceedings of the International Conference on Very Large Databases*, pages 126–135, August 1997.
15. P. Valduriez. Join indices. *ACM Transactions on Database Systems*, 12(2):218–246, June 1987.
16. J. Yang, K. Karlapalem, and Q. Li. Algorithms for materialized view design in data warehousing environment. *Proceedings of the International Conference on Very Large Databases*, pages 136–145, August 1997.

View Derivation Graph with Edge Fitting for Adaptive Data Warehousing *

Ioana Stanoi, Divyakant Agrawal, Amr El Abbadi

Department of Computer Science
University of California Santa Barbara, CA 93106
{ioana,agrawal,amr@cs.ucsb.edu}

Abstract. In this paper we propose adaptive data warehouse maintenance, based on the optimistic partial replication of base source data that has already been used in deriving view tuples. Our method reduces local computation at the view as well as communication with the outside sources, and lowers the execution load on the base sources, which leads to a more up-to-date state of the data warehouse view.

1 Introduction

Data Warehouses integrate and summarize information from one or several base sources in a separate storage, the materialized view. User queries are directed to the materialized view for fast response time, while updates are sent to the base sources and eventually propagated to the materialized view to avoid staleness of the data. After initialization, the computation at the materialized view executes two different tasks. First, the maintenance module refreshes the warehouse view by either re-deriving the whole view, or by incrementally computing and integrating the source updates with the current view [7, 11, 6, 1]. As base sources can be distributed, an important issue during view maintenance is to reduce the communication between the view and the underlying sources. Note that the update effects on the view may be related not only to the base source changes, but also to the current state of other sources. Several update queries are then necessary to gather all this information.

Second, based on the data already stored at the materialized view, user queries are computed and answered. As a result, the choice of a maintenance approach directly affects the speed and the freshness of answers to user queries. Several maintenance approaches for materialized views have been proposed, ranging from full replication of the base source data at the view to the materialization of only the query result tuples. Within this range, self-maintenance algorithms [3, 5] use key and referential integrity constraints to derive a subset of the base source data and maintain a select-project-join view without accessing the outside sources. These methods derive maintenance expressions and eliminate the

* This work was supported in part by NSF under grant numbers CCR-9712108 and EIA-9818320 and IIS98-17432 and IIS99-70700.

Y. Kambayashi, M. Mohania, and A M. Tjoa (Eds.): DaWaK 2000, LNCS 1874, pp. 67–76, 2000.
© Springer-Verlag Berlin Heidelberg 2000

ones that cannot lead to changes at the view. However, this is a pessimistic approach that requires the storage of all the data that can possibly be useful for the view's maintenance. Other methods log a history of the updates or auxiliary information on certain intermediate results [10, 2]. These cases can be considered as pessimistic methods, since they store all the data that can possibly be useful. This can lead not only to a significant storage overhead, but, more importantly, a large volume of data may delay access to the auxiliary view information during maintenance.

By contrast, the *edge fitting* solution we develop in this paper uses the optimistic replication of a subset of information from base sources. That is, we propose to store from each base source only the information already accessed during previous view maintenance, and no computational overhead is needed to derive it. Our optimistic method that reduces the amount of auxiliary information with respect to pessimistic approaches is driven by the necessity to speed up the access to this information and therefore the maintenance task of of the view. Since we reduce the probability of accessing base sources during view maintenance, the sites storing base information can execute more user updates/queries rather than provide view update computation and data.

In Section 2 we present the data warehouse maintenance problem in a distributed system. We further develop the notion of a View Derivation Graph (Section 3) and describe a technique called *edge fitting* for incremental maintenance of data warehouses. In Section 4, we present the details of the proposed algorithm for incremental view maintenance. In the conclusion (Section 5) we present our direction for future work.

2 The Model

A distributed data warehouse model consists of one or more independent views defined over multiple base sources. Communication between two sites is assumed to be both FIFO and reliable. Views derive data from base sources by using SPJ operations as well as an aggregate function g where $g \in \{COUNT, SUM\}$. Note that this set of aggregate functions is easily implemented by storing only a limited amount of information related to the edges of the view derivation graph structure that we propose in this paper. To support an extended set $g \in \{MIN, MAX, AVG, COUNT, SUM\}$ we need access to base source information, procedure whose analysis is beyond the goal of this paper. An attribute is preserved at the warehouse view V if it is either a regular attribute or an aggregate attribute in the view's schema. Projections and selection conditions involving attributes of a single source table are pushed down to the corresponding source. Also, for the purpose of this paper, we assume a tree ordering of the join conditions between different tables, without cycles or self-joins. However, our method supports full, right or left outerjoin. Our algorithm therefore handles a large set of view definitions that in practice are based on the star or snowflake schemas. For simplicity, in some of our examples we describe the computation over the base sources simply as the join operation.

A data warehouse should be initialized, and then maintained up to date with respect to the underlying source data. Let a view V be defined over base sources $B_1, B_2, \cdots, B_n$ as $V = B_1 \bowtie B_2 \bowtie \cdots \bowtie B_n$. An update ΔB_i is propagate from source B_i to V. Then the maintenance task of the view is to compute the effects of ΔB_i by undergoing an *update phase*. An update at V is a function of the change ΔB_i, as well as of the data stored at the base sources. The necessity for communication with several distributed base sources during view maintenance may introduce update delays and lead to staleness of data. Moreover, since base sources both receive and integrate their own updates and answer update queries of the dependent data warehouse views, the communication with the base sources can become a bottleneck in the system. It is therefore essential to reduce not only the number of update queries but also the number of the base sources involved in an update phase. The challenge we are addressing in this paper is therefore twofold: to reduce communication, i.e., reduce the possibility of the view maintenance task being delayed due to source unavailability/failures as well to reduce the amount of storage necessary at the data warehouse. We believe therefore that it is essential to develop **adaptive** maintenance methods that optimistically store only the information that is expected with high probability to be necessary for future updates. Although such a method does not eliminate completely the necessity for communication with the outside sources, it can greatly reduce it.

In this paper we present *edge fitting*, a graph-based method that stores information related to the *path* that leads to the materialization of tuples, i.e., attribute values that lead to the generation of tuples in the warehouse view. Although we present a graphical representation of dependencies, the adaptive maintenance method is not dependent on the structure of this auxiliary information. For example, the view site may store for example in relational tables subsets of base source data previously used for the derivation of the view. In this case, the computation overhead of the join operation is still reduced, due to the lower number of tuples stored. Storing the graph imposes the data structure overhead, but the necessary tuples are accessed faster.

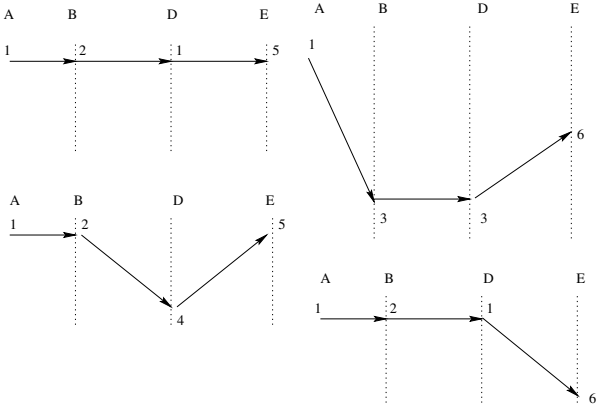
3 The Edge Fitting Method

The point of this paper is not *how* auxiliary data is used, but *what* information we can store without imposing additional computation overhead. We propose the *edge fitting method*, that reuses only the base data that the warehouse view accessed and/or computed during initialization and previous maintenance steps. We further show how this data can be easily used to improve the efficiency of the future maintenance task. Assume a source update is the insertion of a tuple whose attribute values involved in SPJ conditions are already part of the auxiliary data, i.e., have been used in deriving view information. Then the update is simply fit with the previously computed derivation paths, and the final result corresponding to the data stored by a materialized view is changed. If only part of the attribute values of the new update match the auxiliary path information,

then communication is needed to calculate the effect of the remaining attributes. However, if the derivation paths make up a dense graph, this subset is very small, and the probability of communicating with a source is much lower than in the case of the basic incremental view maintenance approach with no auxiliary data.

3.1 Structuring the View Data with Derivation Graphs

To illustrate the **edge fitting** method and the following definitions, consider the following example. A data warehouse view derives information from base source tables B_1, B_2 and B_3, with attributes (A,B), (B,C,D) and (D,E) respectively. We show all tuples stored at the base relations in Figure1. View V is defined as $\prod_{AE} B_1 \bowtie B_2 \bowtie B_3$, a set of tuples (A,E), where A and E are the attributes preserved at the view V. Let a view V derive data from base sources $B_1, \cdots, B_i, \cdots, B_j, \cdots, B_n$. Without loss of generality, assume that the index of a base source corresponds to the order of execution of the view derivation. Then we denote a **node relationship** to be $n_{B_i,B_{i+1}}(v_i) = \{v_{i+1} | (tuple\ \langle v_i, \cdots, v_{i+1} \rangle \in B_i) \wedge (tuple\ \langle v_{i+1} \cdots \rangle \in B_{i+1})$ where v_i is a value of the primary attribute of B_i and v_{i+1} is a value of the join attribute of B_i and $B_{i+1}\}$. If the set $n_{B_i,B_{i+1}}(v_i)$ is null, then there is no join operation that directly relates tables B_i and B_j. Then $n_{B_1,B_2}(1) = \{2, 3\}$ where the primary attribute is A, and the joinable attribute is B; $n_{B_1,B_2}(2) = \{4\}$; $n_{B_1,B_2}(3) = \{4, 2\}$; $n_{B_1,B_2}(5) = \{\}$; $n_{B_2,B_3}(2) = \{1, 4\}$ where the primary attribute is B and the join attribute is D; $n_{B_2,B_3}(4) = \{4, 3\}$; $n_{B_2,B_3}(3) = \{3\}$.



B1	B2	B3
AB	BCD	DE
(1,2)	(2,4,1)	(1,5)
(5,6)	(4,2,4)	(4,5)
(2,4)	(2,1,4)	(1,6)
(3,4)	(4,4,3)	(5,7)
(1,3)	(3,3,3)	(3,6)
(3,2)	(2,3,1)	(5,1)
(6,6)		

Fig. 1. View Derivation Paths for $DP(v_1 = 1)$

The concept of *view derivation paths* expresses the joinability across all sources. A view derivation path is a graphical illustration of joinability instances only across tuples at the base sources that useful in the generation of tuples in the materialized view. Let $B_1 \cdots B_n$ be the base sources accessed by V and indexed according to the execution order during the instantiation of V.

Definition 1 *A* **View Derivation Path** $DP(v_1)$ *for a value* v_1 *of the primary attribute of* B_1 *is a graph* $G(V, E)$ *with* V *as a set of vertices and* E *a set of edges defined as follows:*

1. *An edge* $e \in E$ *represents the same-tuple relationship between two attributes.*
2. *Nodes express the joinability of tuples, and correspond to values of join attributes between tuples. A node has only outgoing edges if it corresponds to a primary attribute, and it has only incoming edges if it represents a view preserved attribute. In general, a node* v_i *corresponds to the value of the join attribute between tuples represented by the incoming and outgoing edges.*

Figure 1 represents the different *View Derivation Paths* corresponding to $DP(v_1 = 1)$, for primary attribute A, join attributes B, D and view preserved attributes A, E.

Finally, we introduce the *derivation graph*, which traces all derivation paths from the primary to all the preserved attributes. Assuming that common edges are indistinguishable, which is a valid assumption for a large set of applications, then multiple common paths are collapsed into single paths. The information stored at the view is used for data warehouse queries, and therefore the view tuples should be readily available. In order to avoid an exhaustive search on the derivation paths, we add to the view derivation graph a set of direct edges between the primary and the view preserved attributes (in the example attributes A and E respectively).

Definition 2 View Derivation Graph
$DG = \bigcup_{v_1 \in PrimaryAttribute(B_1)} DP(v_1)$ *where values* v_1 *are attribute values for the primary attribute of* B_1*. An edge* $\langle v_1, v_{n+1} \rangle$ *exists if there is a derivation path* $DP(v_1)$ *from* v_1 *to the last (according to the base source indexing order) view preserved attribute.*

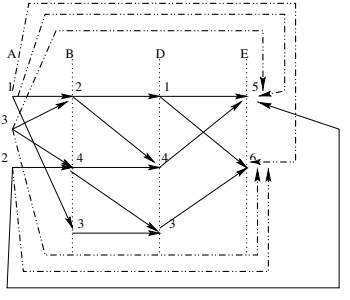


Fig. 2. Derivation Graph before the new updates

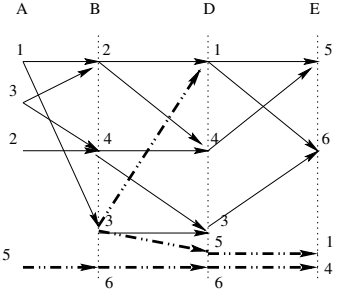
The derivation graph (Figure 2) corresponding to our example, represents all the possible view derivation paths from B_1 to B_n. Nodes are join attribute

values of the different base relations, (B, D), together with the primary (and view preserved) attribute of B_1 and the view preserved attribute of B_3, i.e., (A, E). An edge represents a join pair between two relations.

Next, we will show the effects of an update on a materialized view. We assume that an update is either an insert or a delete operation. Modifications to a tuple in the data sources are propagated at the view as a delete followed by an insert operation.

3.2 Integrating Updates: Inserts

Base sources can be updated, and as a result the derivation graphs should reflect this changes. To continue our example, assume that source B_2 incorporates, consecutively, the updates shown in Figure 3. Figure 3 also represents the derivation graph after the view integrates these new updates. For clarity, we omit the direct edges ($\langle v_1, v_{n+1} \rangle$) between the primary and measure attributes.



B1	B2	B3
AB	BCD	DE
	insert(3,2,1)	
	insert(3,2,5)	
	insert(6,3,6)	

Fig. 3. Update Derivation Graph after the new updates

In general, an update tuple is of the form $\langle v_1, \cdots v_i \cdots \rangle$. For each join attribute value v_i we can consider however one by one the reduced tuple $\langle v_1, v_i \rangle$. Then let an update to a base source B_i have a value $left_{value}$ of the attribute joinable with B_{i-1} and another attribute value $right_{value}$ joinable with source B_{i+1}. For our example, if $\Delta B_i = \Delta B_2 =$ **(3,2,1)**, then $left_{value} = 3$ and $right_{value} = 1$. If $\Delta B_i = \Delta B_2 =$ **(3,2,5)**, $left_{value} = 3$ and $right_{value} = 5$, while if $\Delta B_i = \Delta B_2 =$**(6,3,6)**, then $left_{value} = 6$ and $right_{value} = 6$. An update falls in one of the following three cases:

Case 1: $(left_{value} \in DG) \wedge (right_{value} \in DG)$. The insertion of the first update ΔB_2 = (3,2,1), generates a single edge in the graph (Figure 3). This is because the join attribute values of ΔB_2 have already participated in the generation of tuples at the materialized view. In general, the insertion of an edge in this case either generates one or no new edge in the graph. Communication with the external sources is not necessary.

Case 2: $(left_{value} \in DG) \vee (right_{value} \in DG)$. The second update, ΔB_2 = (3,2,5), illustrates the case when only one of the join attribute values, B, is part

of the previously stored derivation graph. Since the value of the attribute D does not correspond to any value previously used in the join with tuples in B_3, computation is needed for the generation of the remaining derivation path. Note that, if the number of base sources involved in the derivation of a view is large, then there may also be the case that the portion of the path to be added to the derivation graph also merges with the previously stored graph at some attribute node. We distinguish between two cases. If $left_{value} \in$ DG, then base sources $\{B_1, B_2, \cdots, B_{i-1}\}$ need not be queried. If $right_{value} \in$ DG, then the update query should not include sources $\{B_{i+1}, B_{i+2}, \cdots, \}$.
Case 3: $(left_{value} \notin DG) \wedge (right_{value} \notin DG)$. Finally, this is the case where none of the joinable attributes values of the new update (ΔB_3 = (6,3,6)) are part of the derivation graph. Then the effects of the change at B_3 on the materialized view need to be calculated by joining it with the base relation to the right and then to the left of B_3. Note again that, if the derivation graph is dense, then the path to be computed may merge at some point with a path already present in the graph and only a subset of queries are necessary.

3.3 Integrating Deletes

Deleting a tuple $\langle v_1, \cdots v_i \cdots \rangle$ in one of the underlying base relations is interpreted at the view as the deletion of an edge in the graph for each $\langle v_1, v_i \rangle$, v_i being a value of a join attribute. However, if an edge also corresponds to at least one other tuple in the base relation, it should not be deleted. Consequently, there is a need to compute and store *vector tuple counts* as well as *node tuple counts* [11] at each node in the derivation graph. At a node corresponding to some attribute value v_i, the *node tuple count* reflects the derived tuple count, the number of tuples resulting from joining base relations up to B_i, and whose derivation paths go through the node v_i.

Since the resulting tuple can be reached through different derivation paths, each node should store a *vector count* [1] such that each vector entry represents the counter for a given value of the primary attribute. The number of entries in a vector is the number of different values of the primary attribute, in our example attribute A. The following rule is used to initialize and maintain the node counters of a derivation graph, considering the derivation paths from left to right: *the tuple count of a node is the sum of the vectors on incoming edges, multiplied by the corresponding edge tuple count.*

The propagation of a delete operation at the view leads to the deletion of edges that are not used by other derivation paths. In the best case, no edge is actually deleted, but the derivation path is followed and the counters associated with the path are modified. Counters are also modified in the case of a delete operation that leads to a partial, but not full deletion of the derivation path. For example, let the vector count associated with each node have the following entries: ($v_1 = 1$,$v_1 = 3$, $v_1 = 2$). The resulting counters are shown in Figure 4. During the view maintenance, insert and delete updates in the derivation graph

[1] Note that we introduce the notion of vector counts solely for exposition purposes. In the full description of the algorithm, we will eliminate the need for vector counts.

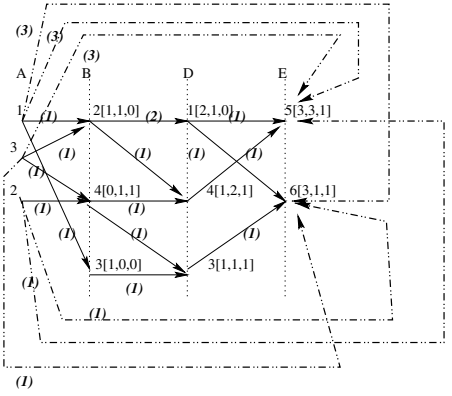


Fig. 4. Vector counter associated with every node in the Update Derivation Graph before updates

will obviously affect some of the tuple counters. Following the completion of the update queries triggered by the changes at source B_2, the derivation graph and the associated node vector counters will be as illustrated in Figure 5 (for clarity, again we omit the direct edges between attributes A and E).

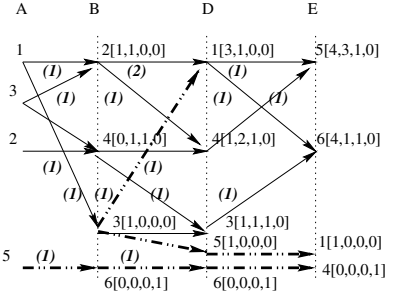


Fig. 5. Vector counters associated with every node in the Update Derivation Graph

4 Discussion on the View Maintenance Algorithm

In this section we give an overview of the *edge fitting* maintenance method. Due to the lack of space, some details that can be found in [8] are avoided.

If the **edge fitting** maintenance method is used, then most of the information of the view derivation graph can be hidden from users, while queries are answered based solely on the information stored by the nodes corresponding to the primary and measure attribute, and the direct edges between them. For both maintenance and query computations, we enforce an index structure for

each joinable attribute, as well as for the primary attribute of the first joinable relation, and the measure attribute. However, for maintenance, the storage overhead for additional information such as the vector counts can become inefficiently large. As the number of attribute values changes, the size of a node vector counter should also be dynamically changed. Although inefficient from a storage point of view, the vector counts associated with the node in the derivation graph provide information on tuple counts at all the points on a derivation path.

Consider eliminating the vector counts associated with every node in the derivation graph. Since the vector counts of the measure attribute values are needed for view queries, the corresponding information can be stored on the direct edges between the primary and the measure attributes.

If the node vector counts are not maintained explicitly, their computation is necessary during view update. For comparison, consider again the derivation graph with vector counts. The insertion or deletion of an edge $\langle left_{value}, right_{value} \rangle$ affects the number of tuples in join operations that involve the node corresponding to $right_{value}$. That is, the vector counts on the derivation paths going though the node $right_{value}$, and related to nodes following $right_{value}$ and up to v_{n+1} are modified. In the case where the insert/delete operation at the view requires the insertion/deletion of more than one edge, the same observation holds for each modified edge. In contrast, a view derivation graph without node vector counts does not offer the information on how many tuples are actually affected by the insert/delete of an edge, and this information needs explicit computation. This can be done by first traversing to the left the derivation paths that contain the node $left_{value}$, computing the total number of tuples affected by the insertion/deletion of the edge, and then traversing all the paths containing the node $right_{value}$. More details on how an insert/delete operation affect view derivation graph without node vector counts, are provided in [8].

5 Conclusion and Future Work

In this paper, we have presented a new approach for incremental maintenance of warehouse views. In order to support adaptive maintenance, we have developed a notion of **view derivation graph** that maintains intermediate information for computing warehousing views and have developed the **edge fitting** algorithm that allows, in the best case, to refresh the views locally as a result of updates to the underlying data sources. In particular, if the data has skewed distribution, which has been increasingly demonstrated for Web data, view derivation graph with edge fitting will have significant benefits. A proposal of an analytical tool to evaluate the suitability of the edge fitting approach for a given application data can be found in [8].

The trends in technology today lead to a partially disconnected world, where data can be generated on portable devices. Traditional approaches for incremental view maintenance rely on executing view computation queries on-the-fly by contacting base sources where data resides. With increasing reliance on warehousing and OLAP it has become important to provide data warehousing

support for non-traditional applications such as mobile and disconnected environments, and unreliable systems. We therefore believe that solutions to the data warehouse maintenance problem need to be developed in the context of a disconnectable environment.

The model presented is developed specifically for one or many independent data warehouse views that derive data directly from base sources. The distributed multi-view data warehouse is emerging as a design solution for a growing set of applications, such as storage of messages from newsgroups [4] or cooperative web caches for the Internet. However, a more general data warehouse system allows views to derive data from other views [9, 6] as well as base sources. We propose to increase the capability of the model to also handle the hierarchical warehouse system.

References

[1] D. Agrawal, A. El Abbadi, A. Singh, and T. Yurek. Efficient View Maintenance in Data Warehouses. In *Proceedings of the 1997 ACM International Conference on Management of Data*, pages 417–427, May 1997.

[2] Y. Cui and J. Widom. Practical Lineage Tracing in Data Warehouses. Technical report, Department of Computer Science, Stanford University, 1999.

[3] Ashish Gupta, H.V. Jagadish, and Inderpal Singh Mumick. Data Integration using Self-Maintainable Views. In *Proceedings of the International Conference on Extending Data Base Theory*, 1996.

[4] H. Gupta and D. Srivastava. The Data Warehouse of Newsgroups. In *Proceedings of the Seventh International Conference on Database Theory (ICDT)*, pages 471–488, 1999.

[5] N. Huyn. Multiple-View Self-Maintenance in Data Warehousing Environments. In *Proceedings of the International Conference on Very Large Data Bases*, 1997.

[6] W. Labio, R. Yerneni, and H. Garcia-Molina. Shrinking the Warehouse Update Window. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, May 1999.

[7] I. S. Mumick, D. Quass, and B. S. Mumick. Maintenance of Data Cubes and Summary Tables in a Warehouse. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, May 1997.

[8] I. Stanoi, D. Agrawal, and A. El Abbadi. View derivation graph with edge fitting for adaptive data warehousing. Technical report, Department of Computer Science, University of California at Santa Barbara, 2000.

[9] I. Stanoi, D. Agrawal, and A. El Abbadi. Modeling and Maintaining Multi-View Data Warehouses. In *Proceedings of the 18th International Conference on Conceptual Modeling (ER'99)*, November 1999.

[10] J. Yang and J. Widom. Temporal View Self-Maintenance in a Warehousing Environment. Technical report, Department of Computer Science, Stanford University, 1999.

[11] Yue Zhuge, Hector Garcia-Molina, Joachim Hammer, and Jennifer Widom. View Maintenance in a Warehousing Environment. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 316–327, May 1995.

On the Importance of Tuning in Incremental View Maintenance: An Experience Case Study (Extended Abstract)*

K. O'Gorman, D. Agrawal, and A. El Abbadi

Department of Computer Science
University of California
Santa Barbara, CA 93106
{kogorman,agrawal,amr}@cs.ucsb.edu

Abstract. We describe the tuning of a gigabyte-scale TPC-D database for investigation of incremental maintenance of a materialized view. We find that incremental maintenance is feasible over a wide range of update sizes (*granularities*), that intuitive SQL formulations perform poorly, but that there are better alternatives. We show that these results can be meaningfully evaluated and presented only in the context of reasonable instance tuning.

Keywords:

View Maintenance, Query and Update Tuning, Distributed Query Optimization.

1 Introduction

Algorithms and frameworks for handling incremental maintenance of materialized views in a data warehouse have been presented largely in the abstract, or tested on small databases[LYC+99]. This work presents the results of exploring (part of) the solution space for implementing incremental maintenance of a view in a data warehouse with a database population of an interesting size (nominally 1 gigabyte).

The tuning research results we are aware of focus on individual elements or operations to be optimized, often using techniques or features that are unavailable in commercial DBMS engines. Fortunately, the importance of tuning a database has long been known to the people who operate commercial databases, and there is an extensive literature available from database vendors (e.g. [Ora], [ALS97]) and third parties (e.g. [GC96,Ada99]). One such source with collaboration from a number of vendors is [Sha92].

While they can be helpful, these volumes are on the one hand overwhelming in their massive content, and on the other hand are most useful in an operational

* This work is partially supported by the NSF under grant numbers EIA-9818320, and IIS-9970700

Y. Kambayashi, M. Mohania, and A M. Tjoa (Eds.): DaWaK 2000, LNCS 1874, pp. 77–82, 2000.
© Springer-Verlag Berlin Heidelberg 2000

context rather than a research context. Blindly following the procedures and advice in these sources could lead to tuning the system specifically to the research tasks, but the goal is to explore performance in a realistic setting. Our approach has been to try for a "middle way" of moderate aggressiveness in tuning, but avoiding distortions to the computing environment.

2 The Data and the Experimental Setting

In this experimental setting one of the key components is the underlying data set that will be used to evaluate view maintenance strategies. We chose the schema of the TPC-D family of benchmarks which has the advantages that it is well-known in the DBMS research arena and that it comes with a generator program dbgen, and a prescribed set of queries. The generator can be used to populate the schema and also to create *refresh sets* which are updates comprising both insertions and deletions. We modified the updates to generate changes to our view, as described in the full version of this paper[OAE00].

All tests were performed using Oracle 8*i* Enterprise Edition, version 8.1.5.0.1, on RedHat Linux Version 6.1, on a single system with a Pentium-III 550MHz CPU. Tests were run by execution of SQL*Plus scripts, sometimes for days at a time.

All tests reported here were obtained with the same layout of partitions and tablespaces; moreover, identical or functionally similar tables were allocated to the same tablespaces in all tests reported here; different allocations were used initially and were helpful in validating the correctness of the scripts.

We chose a view which was a nearly complete denormalization of the schema, in effect precomputing the joins required by several of the prescribed queries (numbers 7, 8 and 9). In building this view we encountered one consequence of the large size of the database: if even a single stage of the query plan has exponential complexity, the computation is infeasible. By various methods, mostly by providing statistics and hints to the optimizer, we got the time down from failure after 48 hours to success after 15 hours, using a query plan based mostly on hash joins.

We verified that queries 7, 8 and 9, when recoded to use the materialized view, increased in speed by factors of 603, 18 and 14.

2.1 Update Granularity

We measured results for five updates of 2400 rows each. These were computed in increments (*grains*) of multiples of 25 rows: 25, 50, 75, 100, 150, 200, 300, 400, 600, 800, 1200 and 2400 orders each. The tests were organized in such a way that the same five results were being computed at each granularity, and only the grain size and the number of grains were changing. This range was an interesting one since it seemed likely at one end that the view tuples being changed would fit in main memory at fine granularity and would not fit at the coarsest granularity. The range from 0.002% to 0.2% of the rows being updated by each grain, seemed reasonable for warehouse updates.

3 Incremental Maintenance of the View

The calculation of the delta for each update used the algorithm from the Posse [OAE99] framework. The calculation proceeds as a series of semijoins, and is followed by installation of the computed change into the view. Installation was explored with SQL and with a cursor-based algorithm. Times are pictured in Figure 1.

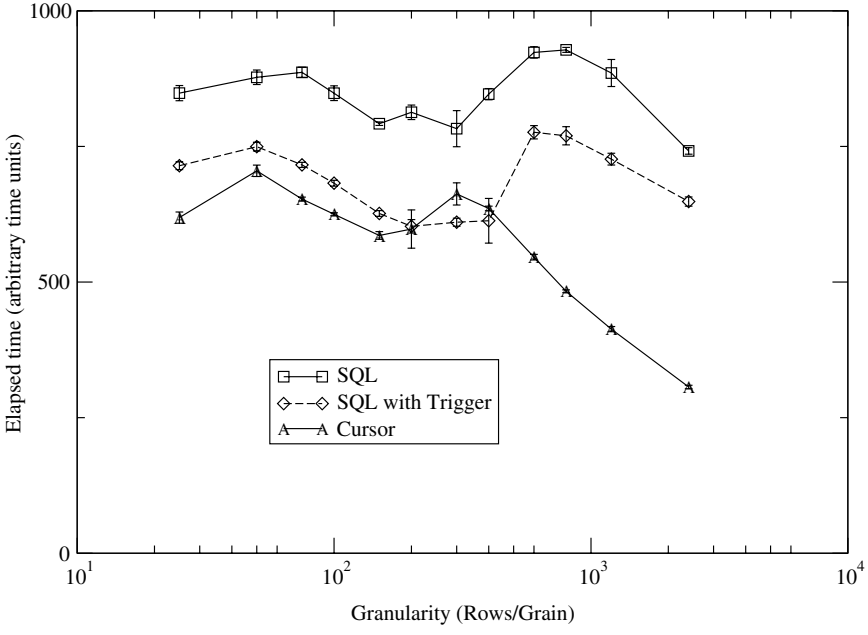


Fig. 1. Execution times with 95% confidence intervals.

3.1 Using SQL

The installation of the computed delta was very slow on our first attempts with SQL. The installation was performed in three consecutive SQL commands: an UPDATE of rows already in the view, an INSERT of new rows, and a DELETE of rows no longer needed. The INSERT had acceptable performance from the start, but the UPDATE and DELETE were an order of magnitude slower.

Tuning an Update. The original form of the UPDATE was the one corresponding to the English description "update rows in the view which correspond to rows in the delta by adding the values in the delta". This works out to a SQL coding style called "coordinated subqueries". Following is an outline of the query:

```
update WARE set WARE.value = WARE.value +
    (select DELTA.value from DELTA where DELTA.<key> = WARE.<key>)
where exists (select 'x' from DELTA where DELTA.<key>=WARE.<key>);
```

In spite of many trials, we could not make the optimizer generate a query plan that drives the update by the key on the delta. Instead, the update used a plan that tested each row of the view for a matching row in the delta, and altered the ones with a match. All of the nearly 6 million rows of the view are read to perform this operation, and the elapsed time is large in proportion to the size of the view table. Eventually, we even found that this behavior is documented for Oracle, but we understand that this is the case in at least some other DBMS engines as well.

Eventually, a suggestion from USENET[Lew99] led us to a much better idea: performing an update on a join, in which the optimizer is free to choose the better table to drive the join. This improved the speed of the update by an order of magnitude because the cardinality of the view is at least 1,000 times that of the delta, and for fine-granularity updates, this difference is even more. As modified, the update is as follows:

```
update (select DELTA.*, WARE.* from DELTA, WARE
    where DELTA.<key> = WARE.<key>)
set <changes>;
```

The performance of this form is essentially that of a physical index read and a row update for each affected row of the view, because the uniform distribution of the data makes it unlikely that any two rows will share a database block. The point here is that had we not discovered this way to tune the SQL, subsequent comparisons would have been invalid because the SQL would have been unnecessarily impaired.

Tuning a Delete. For a view that contains statistical data, it would be feasible to omit the deletion step entirely, and allow tuples to remain that contain only zeroes outside the identifying attributes. We chose to include and measure the deletion step, but tuned the SQL.

Since there is no way in SQL to specify conditional deletion in connection with an update, deletion has to be a separate step. In order to unambiguously identify rows to be deleted, the view contains an attribute count that encodes the number of rows of the LINEITEM table that contribute to the entry in the view. A row can be deleted when this value goes to zero.

Tuning of deletion required considering three ways of formulating the SQL. The obvious way is to test each tuple for a count of zero, but this requires a complete scan of the view. A faster way is to use an index on the count attribute, at the cost of maintaining the index, but with the advantage that only the tuples to be deleted (i.e. those with count equal to zero) will be accessed. A third way

involving triggers will be described below. Since only tuples that have just been updated by the delta will have changed, just these tuples can be examined. Formulating efficient SQL for this is a challenge similar to the one for tuning the SQL of the update, and results in a formulation like the following[1], due again to help from USENET[Lew99]:

```
delete from WARE ww where (ww.<WARE key attributes>)
    in (select dd.<DELTA key attributes> from DELTA dd);
```

Triggers. These SQL versions used an index on the count attribute to support deletion, and to see whether removing this requirement would close the performance gap with the cursor-based version, the tests were re-run on a version that used a trigger to maintain a table of rows to be deleted. This improved performance considerably.

3.2 Using a Cursor

No matter how much we tuned the individual SQL commands, they remained three distinct commands, and the optimizer does not apply any global optimization to take advantage of the fact that all three are working with the same data. In order to do that, we were obliged to code a query plan that SQL cannot accomplish, and to do that we used a cursor-based approach. The idea is simple: the process will proceed through the delta one row at a time and make whatever changes are necessary to the view for that row. The resulting cursor-based code was faster, by about a factor of two, in all tests. The code for cursors, omitted for reasons of space, is included in the full version of this paper[OAE00].

4 Conclusion

We have presented our experience with performing timing testing of view maintenance for a warehouse on a commercial DBMS, and some of the pitfalls for the unwary.

- Tuning is essential to arriving at correct conclusions about algorithm performance; as a corollary, reporting on tuning-related aspects of the computing environment is essential for others' evaluation of reported results.
- Comparing our tuned algorithms, at some granularities we find that cursor-based updating may be as much as four times faster than SQL with an index-driven DELETE, and somewhat faster than a trigger-based SQL solution.

[1] Actually, hints to the optimizer are also required, but are omitted because they are not portable. Of course, on different systems it could be that any of these formulations would work differently or require additional or different hints to the optimizer. Portability of SQL really is an illusion.

The untuned versions were sometimes so bad we were unwilling or unable to complete timing tests. We conclude that comparing untuned algorithms is meaningless, and running them is not sensible.
- A version using cursors was the clear winner among algorithms for incremental maintenance. This algorithm makes use of global properties of the computation and cannot be produced by optimizers which operate one SQL statement at a time.
- We saw improvements from a number of tuning techniques. There was no clear winner. One surprise was that the most efficient SQL for a given task may be quite different from the first formulation that comes to mind, although they may be theoretically equivalent.
- Incremental view maintenance appears feasible at all the grain sizes we tested, although naturally larger grains lead to longer periods with locked data. The overall time was surprisingly uniform; it appears that economies of scale were offset by cache effects to some extent. We conclude that where other considerations allow, the short update window that goes with small update granularities may be attractive.
- Incremental maintenance is at least an order of magnitude faster than complete recomputation of the view. At one gigabyte of data, complete recomputation is not feasible on more than a very occasional basis.

References

[Ada99] Steve Adams. *Oracle 8i Internal Services for Waits, Latches, Locks, and Memory.* O'Reilly, 1999.

[ALS97] Eyal Aronoff, Kevin Loney, and Noorali Sonawalla. *Advanced Oracle Tuning and Administration.* Oracle Press, 1997.

[GC96] Mark Gurry and Peter Corrigan. *Oracle Performance Tuning.* O'Reilly, second edition, 1996.

[Lew99] Jonathan Lewis. Usenet postings and private emails. URL http://www.jlcomp.demon.co.uk, 1999.

[LYC+99] W.J. Labio, J. Yang, Y. Cui, H. Garcia-Molina, and J. Widom. Performance issues in incremental warehouse maintenance. Technical report, Stanford, October 1999.

[OAE99] Kevin O'Gorman, Divyakant Agrawal, and Amr El Abbadi. Posse: A framework for optimizing incremental view maintenance at data warehouses. In Mukesh Mohania and A. Min Tjoa, editors, *Data Warehousing and Knowledge Discovery First International Conference, DaWaK '99*, number 1676 in Lecture Notes in Computer Science. Springer, August/September 1999.

[OAE00] Kevin O'Gorman, Divyakant Agrawal, and Amr El Abbadi. On the importance of tuning in incremental view maintenance: An experience case study. Technical Report TRCS00-04, University of California at Santa Barbara, Department of Computer Science, UCSB, Santa Barbara, CA, 92106, March 2000.

[Ora] Oracle Corporation. *Oracle 8i Tuning.* Part No. A67775-01.

[Sha92] Dennis E. Shasha. *Database Tuning: A Principled Approach.* Prentice Hall, 1992.

BEDAWA – A Tool for Generating Sample Data for Data Warehouses

Thanh N. Huynh, Binh T. Nguyen, J. Schiefer, A M. Tjoa

Institute of Software Technology (E188)
Vienna University of Technology
Favoritenstr. 9 - 11 /188, A-1040 Vienna, Austria.
({thanh, binh, js, tjoa}@ifs.tuwien.ac.at)

Abstract. The generation of representative sample data for data warehouses for benchmarking, testing and presentation purposes is a challenging complex task. Difficulties often arise in producing familiar, complete and consistent sample data of any scale. Producing sample data manually often causes problems, which can be avoided by an automatic generation tool producing consistent and statistical plausible data. The data generated by the tool is based on the linear statistical model of a hierarchical n-way classification with fixed effects. We present an approach for a flexible generation of "real world" sample data using the BEDAWA (**BE**nchmarks for **DA**ta **WA**rehouses) tool.

1. Introduction

In the field of data warehouses [2], [4], [11] and OLAP [5], [7], [8] technologies, sample data is needed for different purposes, such as benchmarking, testing, and demonstrating. Unfortunately there exists a lack of tools available to generate sample data, which is familiar, consistent, scalable, and reflective multiple degrees of freedom. Referring to sample data generation problem, [14] analysed the samples for data warehouses and OLAP products of well-known companies, and concluded that the generation of test-data for their products is not yet sufficiently solved.

The sample data generation process is usually a complex, iterative process that begins with a modelling step, and ends with the generation of the desired sample data.

In this paper, we propose BEDAWA –as a *tool to generate generic sample data for data warehouses*. To get representative sample data that is familiar, consistent, scalable, large amount and reflects many degrees of freedom, the BEDAWA tool is based on a statistical model that will be introduced in section 3. The statistical correctness of the model provides a framework to define relationships between dimensions and facts in the context of star schema, the most popular schema for designing and building a data warehouse [10], [12]. Furthermore, BEDAWA provides abilities to define and generate sample data of any size in order to reflect a real world situation. In addition, BEDAWA is also able to integrate existing external data sources for the sample data generation.

The remainder of this paper is organized as follows. In section 2, we discuss related works. Section 3 introduces the statistical model of our prototype. In section 4,

Y. Kambayashi, M. Mohania, and A M. Tjoa (Eds.): DaWaK 2000, LNCS 1874, pp. 83–93, 2000.
© Springer-Verlag Berlin Heidelberg 2000

we present the BEDAWA sample data generation process. A description of the sample data generation process of BEDAWA is presented in section 5. A typical example of generating sample data for Data Warehousing is shown in section 6.The paper concludes with section 7, which presents our current and future works.

2. Related Works

Benchmarks are used for measuring and comparing the operations of systems or applications working on these systems. This especially applies to databases where many efforts have been made to develop tools for benchmarking purposes.

The Wisconsin benchmark [6] has been proved as suitable benchmark for measuring the queries on relational database systems. AS^3AP is another benchmark that tests multi-user models by different types of database workloads [17] for relational databases. The TPC-D benchmark [16], a decision support benchmark was introduced in 1990, with its first version completed in 1995. In 1998, the TPC-R and TPC-H benchmarks [18] were introduced to allow more precise and domain specific performance measurement.

There are many other benchmarks that have been developed for testing and comparing different systems in various application domains such as "The Set Query Benchmark", "The Engineering Database Benchmark", and "Engineering Workstation-Server" [3].

For data warehouses and OLAP, few approaches are known in the literature. In [13], an OLAP benchmark, called APB-1, was issued to measure the overall performance of an OLAP server. Typical OLAP operations are included in this benchmark such as bulk loading of data from internal or external data sources, calculation of new data, time series analysis, and so on.

3. Statistical Model

Need of Statistical Model

The definition of a benchmark can be a very time consuming task. For instance, for defining TPC-A benchmark, TPC required nearly 1200 man-days of effort [15]. This is particularly true for data warehouse and OLAP systems, where the required data volume can be become huge. For business purposes we need a save analysis of the information dependencies of the enterprises to create data for an OLAP system which is consistent with respect to the different granularity levels and the interdependencies between different dimensions. To generate sample data for these systems, a rigorous calculation model which is based on a sound statistical approach is needed. The model does not only allow users to present the relationships between the data elements, but also take into consideration the limitation given by the degrees of

freedom of generated sample data. In the process of generating such large and complex data without the help of an appropriate model, unpredictable consistency errors can appear and they make a tool being useless, i.e., the generated sample data is neither consistent, nor realistic.

The Statistical Model

For simulating realistic OLAP situations, depending on the structure of future sample data, a suitable calculating model can be selected. Various calculating models can be applied on different structures of the sample data with different meaning. We apply a linear statistical model to calculate facts in the context of the star schema [10]. This linear statistical model is applied to present the relations between the fact and the dimensions. Each element of a dimension will have an effect to the fact, called "fact effect value". A fact value of a fact is calculated basing on the fact effect values of elements of the dimensions. Considering to the relationship between the fact and the dimensions, which effect to the fact, the actor who interacts with the BEDAWA tool will define these relationship by using the statistical linear model of -way classification with fixed effects. For simplicity and readability reason we omit the interaction. In the following, we will use in the paper the n-way classified model without interaction.

The linear model, which can be found often in the real world is extremely suitable for presenting the relations between the dimensions and the fact in star schema.

The linear model used in the BEDAWA tool is described as follows [1]:

$$y_{ijk...} = \mu + \alpha_i + \beta_j + \gamma_k + ... + \varepsilon_{ijk...} \qquad (1)$$

where:

- $\alpha, \beta, \gamma, ...$: fact effect value sets of dimensions D_α, D_β, D_γ that have effect on the fact y, the dimensions have numbers of elements in their domains are n, m, p, ... respectively,
- $i, j, k, ...$: index numbers that are in range [1..n], [1..m], [1..p], ..., respectively,
- μ : average mean of the fact,
- $\alpha_i, \beta_j, \gamma_k, ...$: fact effect values of elements i^{th}, j^{th}, k^{th} of dimension fact effect value sets $\alpha, \beta, \gamma, ...$, respectively,
- $\varepsilon_{ijk...}$: error random value, using normal distribution function, $N(0,\sigma^2)$.
- $y_{ijk...}$: a value of the fact.

And the conditions:

$$\sum_{i=1}^{n} \alpha_i = 0 \text{ ,} \qquad \sum_{j=1}^{m} \beta_j = 0 \text{ ,} \qquad \sum_{k=1}^{p} \gamma_k = 0 \text{ , ...} \qquad (2)$$

In this model, we assume that there is no interaction between the fact effect values of each couple of dimensions. That means the fact effect values of a dimension do not depend on the fact effect values of any other dimensions, which influence to the fact.

In context of the hierarchy structure of a dimension, the dimension data can be presented in a tree with different levels of its hierarchy, see figure 1. A branch of this tree presents an element in the dimension domain. Consequently, the fact effect value

of the dimension applied to a fact can be calculated dependent on the effect of all nodes of this branch. For each node of the dimension tree, the node's fact effect value can be defined for the fact. The fact effect value in this context is the effect of the branch from the root to the leaf of the dimension tree. We calculate the fact effect value of a branch by summing all fact effect values of all nodes of that branch. For instance, the fact effect value α_i of the dimension D_α is calculated by the formula:

$$\alpha_i = a_j + a_{j_k} + a_{j_{k_l}} + \ldots \quad (3)$$

where a_j, a_{j_k}, $a_{j_{k_l}}$ ... are the fact effect values of the nodes of the branch α_i.

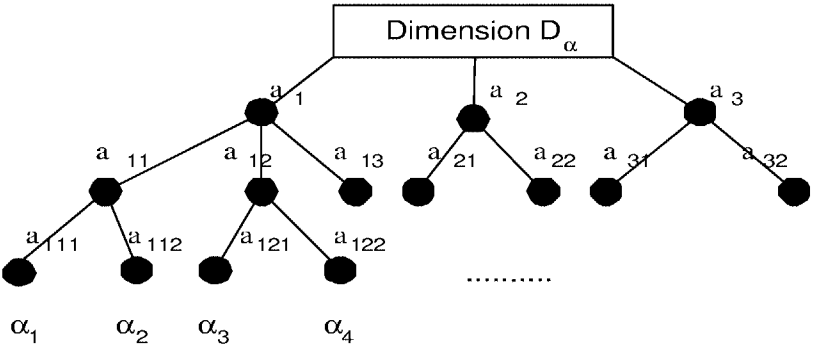


Fig. 1. The fact effect value tree of the dimension D_α

By the condition (2) we can apply the condition that at any node of the dimension tree, sum of all node fact effect values of its sub-nodes is equal to zero:

$$\sum_j a_j = 0, \qquad \sum_k a_{j_k} = 0, \qquad \sum_l a_{j_{k_l}} = 0, \ldots \quad (4)$$

Example

To demonstrate our approach, we take the star schema of well-known grocery example of Kimball [11], see figure 2. The figure 3 shows the relation between the fact *Dollar_Revenue* (y) with 3 dimensions: *ProductDimension* (α), *TimeDimension* (β), and *StoreDimension* (γ) and an average mean (μ).

From the formula (1) and the definition of the fact effect value tree, we have a corollary that one dimension can affect different facts and therefore they have different fact effect value trees. For instance, the fact *Customer_Count* is affected by the dimensions *TimeDimension* and *ProductDimension*. These dimensions can have other fact effect value trees, e.g. α', β', γ', which are used to calculate the *Customer_Count* fact values.

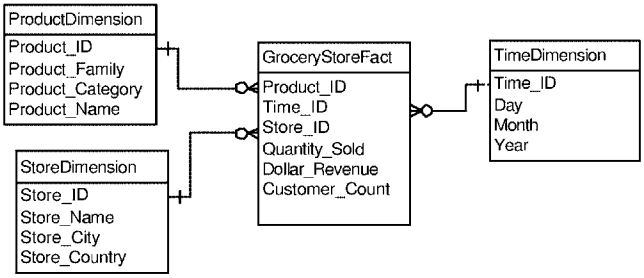


Fig. 2. A star schema of the GroceryStore example

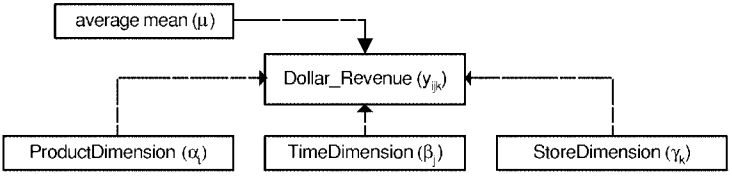


Fig. 3. Effect of dimensions to the fact Dollar_Revenue

4. The Sample Data Generation Process

The sample data generation is an iterative process, which starts with the modelling of the required sample data, and ends with the automatic generation of this data. During this process there are many possible iterations that are dependent on the results of the sample data generation. If the produced data is not satisfying or errors occur during the sample data generation, the design of sample data has to be changed or altered to achieve a better result. An actor can repeat the generation of the sample data, until (s)he receives the expected result. Typically, during the first iterations the user generates smaller amounts of sample data, and after assuring that all requirements are reasonably fulfilled the user can start with the production of other large amounts data.

A sample data generation process may be characterized by three functional areas:

- *The build-time functions*, concerned with defining, and the possibly modelling of sample data requirements and definitions
- *The run-time functions*, concerned with executing the production of sample data
- *The run-time interactions* with human users and applications for controlling the sample data generation

Figure 4 shows the stages for a sample data generation and the relationships between these main functions.

Build-time Functions. Build-time functions are used to build all necessary definitions for the desired sample data. These definitions cover the full description of the sample data being built for the data warehouse.

Run-time Functions. At the run-time stage the data of the repository with the sample data definitions is used to generate the desired sample data. Run-time functions provide the functionality to produce sample data that corresponds to their

definitions. They check the consistency of sample data definitions and ensure that all constraints are satisfied.

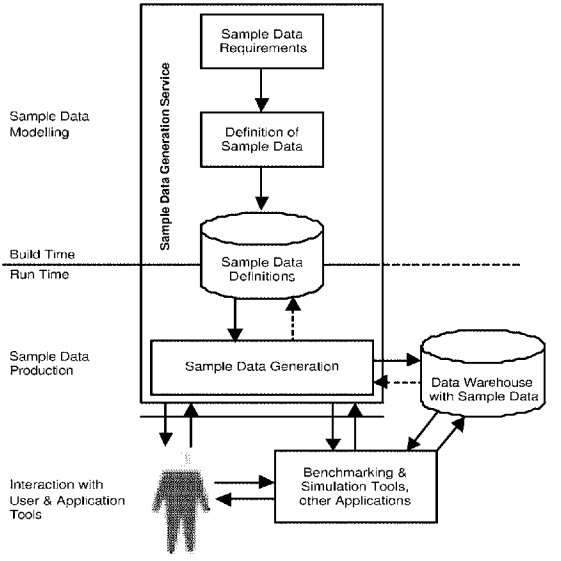


Fig. 4. Sample Data Generation Process

Run-time Interaction Functions. The run-time interaction functions act as links between the run-time functions and the actors, which manage the *sample data generation service (SDGS)* from the outside. Actors can be either human users or applications that are able to control the sample data generation by interactions like executing, cancelling or adjusting the sample data generation. Interaction with the SDGS is necessary to initiate the generation process, to treat exceptions during the sample data production (i.e. by adapting the sample data definitions) or to stop the generation process.

5. BEDAWA Tool

Overview

In BEDAWA project, we implemented a prototype of the previously mentioned sample data generation process. The statistical model introduced in section 3 is implemented in this prototype. Additionally BEDAWA provides functionality to define constraints for the generation process. This additional feature makes it easier

for the user to control the generation process and to check, whether the produced sample data fulfils the requirements modelled in the build-stage.

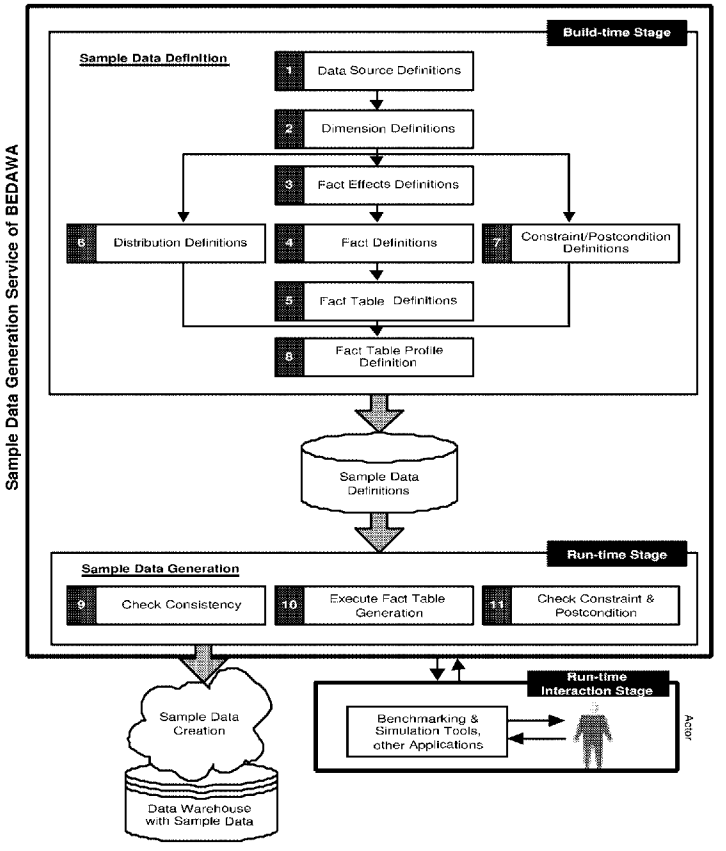


Fig. 5. The Sample Data Generation Process of BEDAWA

Every time sample data is generated for the data warehouse, the following 11 steps are required (see figure 5). Steps 1-8 belong to the build-time stage. This stage covers the modelling and the design of the sample data. In this stage all definitions for the sample data are completed which are required to generate the sample data automatically. The next steps 9-11 belong to the run-time stage and are necessary to generate the modelled sample data. All steps are described in the following section in more detail.

The run-time interaction stage covers the interaction with human users and applications for controlling the sample data generation. That means, depending on the result of the sample data generation in this stage, possible iterations of the presented steps can be necessary. If for instance the result of the sample data production doesn't fulfil the requirements, the actor will have to go back to the build-time stage to model the sample data correctly. The actor iterates between these steps until the produced sample data is satisfying.

The Use of BEDAWA in the Process of Sample Data Generation:

- **Data Source Definitions.** Data source definitions provide flexibility in support of multiple heterogeneous data sources. They hold the connection information to data providers, which can be any supported xternal data sources (i.e. relational database, flat file). Data source definitions are used in our model for two purposes. First, these definitions can be used to define the connection information for retrieving pertinent data for the definition of sample data from a data provider. Second, a data source definition has to indicate, where the generated sample data has to be stored.
- **Dimension Definitions.** Dependent on the characteristics of the modelled sample data, different dimension structures are required. For generating sample data that is used for different purposes on various systems, the BEDAWA tool supports a flexible way to define the dimension structures and is also able to automatically generate data for dimensions. The hierarchy of a dimension can be defined by expected number of levels, and the data type for every level.
- **Distribution Definitions.** The distribution definition is used to specify the distribution of the sample data that simulates real life data. By making statistics on real life data, e.g., statisticians can answer the question, "How many percent of a product was bought? ". In contrary, the generated sample data process uses distribution information to distribute the reference of the dimension elements to a fact table. The distribution information is defined by percent numbers, and has to be provided for every node of a hierarchy level of a referenced dimension.
- **Fact Effect Definitions.** Based on the tree representation of the dimension domain, the fact effect value of every node of the tree has to be defined. For each relation between a dimension and a fact, one or more fact effect value set(s) can be defined. The fact effect values, defined for all nodes of the tree have to satisfy the condition (4).
- **Fact Definitions.** As mentioned, one or more fact effects (α,β,γ) can be used to define a fact for determining their relationship. Furthermore, an average mean (μ), and an error random value (ε_{ijk}) are necessary to specify a fact in this step.
- **Fact Table Definitions.f** The structure of the act table can be defined by specifying a list of facts and dimensions. For example, if the fact table *Sale2000* is defined by specifying the fact *SaleCount* and three dimensions *Geography*, *Product*, and *Time*, then the structure of the fact table *Sale2000* consists of three foreign key columns, e.g. *Geography_ID*, *Product_ID*, and *Time_ID*; and one fact column, named *Sale Count*.
- **Constraints.** Depending on the constrain definitions, the actors can decide how the generated sample data should looked like. At any time in the build-time stage, actors can define constraints for the data of the fact table and boundaries for other generated data. For instances, in the fact definition step, constraints for the fact can be defined, e.g., a fact value of this fact must be in a range [x..y], a fact value must be greater than zero and so forth.

- **Fact Table Profile Definitions.** A fact table profile definition contains all information to generate a fact table including 1) Data source definition, in which the sample data will be stored, 2) a fact table definition is used to create a fact table and to derive a list of fact definitions and dimension definitions. In some case, the name of fact table can be changed in this step, 3) the number of records, which should be generated in the fact table, 4) distribution definitions are used to distribute data for dimension columns in the fact table, and 5) the constraints are used to control the generation process.
- **Sample Data Generation.** Sample data generation consists of three main steps: 1) check consistency, 2) execute fact table generation, and 3) check constraints. After selecting a fact table profile, the generation process begins with checking the consistency of the sample data specification. In the next step IDs are first distributed to dimension_id columns in the fact table. Afterward, fact values are calculated for every fact column. The last step is checking whether the generated data fulfils the constraints or does not. If not, generation step must be re-executed until the generated data meets the requirements of the application.

6. Results

	PRODUCT_ID	STORE_ID	TIME_ID	QUANTITY_SOLD	DOLLAR_REVENUE	C
1	95	53	178	10049	155	
2	35	6	155	10023	131	
3	151	21	95	9957	57	
4	114	11	85	9999	93	
5	189	56	18	9971	71	
6	186	47	164	10007	95	
7	134	27	85	9999	102	
8	195	38	86	9930	36	
9	193	38	32	9940	46	
10	104	57	15	9957	48	
11	189	47	110	9953	59	
12	161	13	41	9993	84	
13	41	23	146	9910	19	
14	157	49	68	9996	93	
15	31	9	6	9939	36	
16	123	60	11	10083	186	
17	84	59	100	10046	134	
18	124	33	14	9909	12	

Fig. 6. The representation of the fact table Grocery (first 18 rows of 10000)

The BEDAWA tool has been used to generate several results of sample data for purposes of benchmarking, testing and demonstrating data warehouses and OLAP systems. In this section we show how to generate sample data for the well-known *GroceryStore* example from [11]. The *GroceryStore* schema is presented in figure 2.

Following the steps, in section 5, we generate GroceryStore fact table with 10000 records, the generated sample data is shown in figure 6. Generally, there is no limitation of the number of generated records using the BEDAWA tool. This ability ensures that the tool can be used to define and generate scalable sample data adapting for the need of various usage purposes.

7. Conclusion and Future Works

We introduced the need of having a tool to generate sample data for data warehouses. In the context of generating sample data, we presented the tool within the BEDAWA project. The BEDAWA tool has a sound statistical foundation by using and implementing the statistical model of an n-way classification with fixed effects. The theoretical soundness of the model provides a framework to define relationship between dimensions and facts in context of the star schema.

The paper presented the process of generating sample data for data warehouses. The process provides abilities to define and to generate sample data. In addition, the tool provides functionality to define constraints to check whether the sample data meets the application requirements or does not.

In conclusion, the implemented prototype is able to flexibly and effectively generate sample data of specific applications. As a result, the generated sample data is statistical correct and "looks like" real business data.

In the future, we will extend the BEDAWA tool to support complex data structures for sample data such as snowflake schema. Further research must be pay attention to extending the adaptability of the statistical model to simulate real life data, that means, more statistical models are need to present various relationships in the generated sample data. Furthermore we are going to offer the BEDAWA tool as a web service. For this purpose we extend our prototype with CORBA technology to allow user unlimited access to the sample data generation engine of the BEDAWA tool. In addition we offer the sample data in XML and DDL (Data Definition Language) to provide a wide application of the generated data.

Acknowledgement

We are very indebted to Microsoft Research (Cambridge, UK) for providing the hardware for the BEDAWA project and for funding the Lab2000 at the Institute of Software Technology for this research project. We have also to thank Professor G.

Vinek who as a statistical expert on experimental design gave us helpful advice for the generation of sample data.

References

[1] R. Christensen, Plane Answers to Complex Questions: The Theory of Linear Models, Springer-Verlag, 1987.

[2] R. C.Barquin, H. A.Edelstein, Planning and Design the Data Warehouse, Prentice Hall PTR, 1997

[3] R. Cattell, The Benchmark Handbook, 1993.

[4] S. Chaudhuri, U. Dayal, An Overview of Data Warehousing and OLAP Technology, SIGMOD Record Volume 26, Number 1, September 1997.

[5] E. Codd, S. Codd, and C. Salley, Providing OLAP (On-Line Analytical Processing) to user-analysts: An IT mandate, Technical report, 1993.

[6] D. J. Dewitt, The Wisconsin Benchmark: Past, Present, and Future, The Benchmark handbook, chapter 4, 1993.

[7] V. Harinarayan, A. Rajaraman, J. D. Ullman, Implementing Data Cubes Efficiently, SIGMOD Conf. 1996: 205-216.

[8] J. M. Hellerstein, The Case for Online Aggregation, SIGMOD 1997.

[9] W.H. Inmon, Building the Data Warehouse, John Wiley, 1992.

[10] R. Kimball, The Data Warehouse Toolkit, John Wiley & sons inc., 1996.

[11] R. Kimball, A Dimensional Modelling Manifesto, DBMS Magazine, 1997.

[12] R. Kimball, L. Reeves, M. Ross, W. Thornthwaite,The Data Warehouse Lifecycle Toolkit, John Wiley & Sons, 1998

[13] OLAP Council, APB-1 OLAP Benchmark. Release II, 1998

[14] J. Schiefer, A. Min Tjoa, Generating Sample Data for Mining and Warehousing, Proc. Int. Conference on Business Information System, Springer-Verlag, 1999.

[15] O. Serlin, The history of debitcredit and the TPC, The Benchmark handbook, chapter 2, 1993.

[16] Transaction Processing Performance Council, TCP-D benchmark.

[17] C. Turbyfill, C. Orji, D. Bitton, AS^3AP- An ANSI SQL Standard Scalable and Portable Benchmark for Relational Database Systems, The Benchmark handbook, chapter 5,1993.

[18] R. Winter, P. Kostamaa, On-the-fly on the up and up, Intelligent Enterprise Magazine, November 16th, 1999.

DyDa: Dynamic Data Warehouse Maintenance in a Fully Concurrent Environment *

Xin Zhang and Elke A. Rundensteiner

xinz | rundenst@cs.wpi.edu
Department of Computer Science, Worcester Polytechnic Institute, Worcester, MA 01609

Abstract. Data warehouses (DW) are an emerging technology to support high-level decision making by gathering information from several distributed information sources (ISs) into one materialized repository. In dynamic environments such as the web, DWs must be maintained in order to stay up-to-date. Recent maintenance algorithms tackle this problem of DW management under *concurrent data updates* (DU), whereas the EVE system is the first to handle (*non-concurrent schema changes*) (SC) of ISs. However, the concurrency of schema changes by different ISs as well as the concurrency of interleaved SC and DU still remain unexplored problems. In this paper, we propose a solution framework called DyDa that successfully addresses both problems. The DyDa framework detects concurrent SCs by the broken query scheme and conflicting concurrent DUs by a local timestamp scheme. The two-layered architecture of the DyDa framework separates the concerns for concurrent DU and concurrent SC handling without imposing any restrictions on the autonomy nor on the concurrent execution of the ISs. This DyDa solution is currently being implemented within the EVE data warehousing system.

Keywords: Data Warehousing, View Maintenance, Data Updates and Schema Changes, Concurrency.

1 Introduction of Data Warehousing

Data warehouses (DW) are built by gathering information from several autonomous Information Sources (ISs) and integrating it into one virtual repository. Data warehousing [6] has importance for many applications in network environments, such as travel services, E-commerce, decision support systems, and other web related applications. An important consequence of the autonomy of sources is the fact that those sources may change without being controlled from a higher data integration layer.

* This work was supported in part by several grants from NSF, namely, the NSF NYI grant #IRI 97–96264, the NSF CISE Instrumentation grant #IRIS 97–29878, and the NSF grant #IIS 97–32897. Dr. Rundensteiner would like to thank our industrial sponsors, in particular, IBM for the IBM partnership award, and GTE for partial support of Xin Zhang.

Y. Kambayashi, M. Mohania, and A M. Tjoa (Eds.): DaWaK 2000, LNCS 1874, pp. 94–103, 2000.
© Springer-Verlag Berlin Heidelberg 2000

1.1 DW Management under Fully Concurrent Environment

Many sources, particularly WWW-based data sources, change not only their data, but also their schema or query interface without cooperating with users of their data. Even conventional databases typically undergo schema changes due to many reasons, including application requirement changes, corrections of schema design errors, upgrades, etc. Case studies both by [11] for health management applications and [5] for different business type of application reveal that schema changes are inevitable not only during the development of a project but also once a project has become operational. Even seemingly stable data sources that appear to be only undergoing data updates may indirectly cause schema changes to be propagated to the DW via these wrappers, e.g, SchemaSQL [3]. Therefore, any data warehouse has to take the possibility of IS schema changes into account.

Furthermore, since ISs are autonomous, schema changes can occur concurrently and can be mixed arbitrarily with data updates from other ISs. Hence, data warehouses built on top of those ISs have to meaningfully handle these updates if their views are to survive over a period of time.

There are three types of tasks related to DW management in distributed environments. The most heavily studied area is the incremental maintenance of materialized views under distributed ISs. Such view maintenance (VM) algorithms [12, 1, 16, 15] maintain the extent of the data warehouse whenever a data update (DU) occurs at the IS space. The second area called view synchronization (VS) [9, 10] is concerned with evolving the view definition itself whenever there is a schema change (SC) of one of the ISs that results in a view definition to become undefined. The third area, referred to as view adaptation (VA) [2, 6, 7], adapts the view extent incrementally after the view definition has been changed either directly by the user or indirectly by a view synchronization module.

Materialized view maintenance (VM) is the only area among those three that thus far has given attention to the problem of *concurrency of (data) updates* at ISs [12, 1, 16, 15]. Our recent work on Coop-SDCC [14] is the first to study the concurrency problem of both data updates and schema changes in such environments. The Coop-SDCC approach integrates existing algorithms designed to address the three DW management subproblems VM, VS and VA into one system by providing a protocol that all ISs must abide by and that as consequence enables them to correctly co-exist and collaborate. In that solution, the SDCC system *locks* the ISs to wait for the mediator to accomplish the maintenance of the data warehouse before admitting a SC to be executed at that IS and thus propagated up to the DW. This solution has however the limitation of requiring information sources to *cooperate* by first announcing an intended SC, then waiting for the DW to finish servicing any outstanding requests, before being permitted to execute the schema update at the IS.

1.2 Our Approach

In this paper, we overcome the limitation of this previous solution. We now propose the Dynamic Data warehouse (DyDa) framework that can handle fully

concurrent SCs and DUs without putting any restriction on the timing of when an SC is allowed to take place nor requiring any cooperation from the ISs.

Since we release the restriction of cooperative ISs, during the process of managing a data warehouse, query requests that are sent down to different ISs [1, 15], new updates (DU or SC) could occur concurrently at some of the ISs. We call such updates *concurrent updates*. A concurrent *DU* will result in an *incorrect query result* returned by an IS [14, 8], whereas as a concurrent SC results in a *broken query* that cannot be processed by the ISs, i.e., an error message.

In this work, we provide the first complete solution called DyDa with:

- We characterize the problem of maintenance under concurrent SCs, which we call the broken-query problem.
- We devise a strategy for the detection of concurrent SCs based on the broken query concept, that identifies the causes of the conflict.
- We analyze selected VM, VS and VA algorithms from the literature to determine if and how they are affected by concurrent SCs, and thus elicit their properties required to function correctly in this dynamic environment.
- We introduce the overall solution framework DyDa that adapts a two-layered architecture that separates the concerns for concurrent DU and concurrent SC handling without imposing any restrictions on the fully concurrent execution of the ISs.

2 Problem Definition

Background on Schema and View Definition Changes. We have to first understand how DW management algorithms, i.e., view maintenance (VM), view synchronization (VS), and view adaptation (VA), work together (see the Figure 1). The VS algorithm will update the view definitions affected by schema changes (SC) at the IS. The output of VS will be a set of view definition changes (VDCs), which are used to update the affected view definitions. Then the VA algorithm will adapt the extent of the rewritten views based on the set of VDCs. At the same time, VM will continuously update the view extent based on data updates (DU) that independently may occur at different ISs.

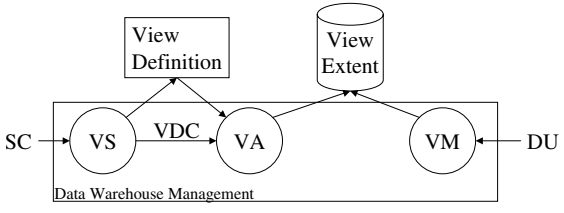


Fig. 1. Relationship between VM, VS and VA Algorithms

Definition of the Maintenance-Concurrent Problem. While the concept of maintenance concurrency has previously been defined for data updates only [12], we now extend it to incorporate schema changes. In the following we assume the views are defined in SPJ queries and the connection between each IS and DW is FIFO.

Figure 2 defines the main notations we use. A sequential number n, unique for each update, will be generated by DyDa whenever the IS update message reaches the system.

Definition 1. *Let $X(n)[j]$ and $Y(m)[i]$ denote either DUs or SCs on IS[j] and IS[i] respectively. We say that the update X(n)[j] is* **maintenance-concurrent** *(in short* **concurrent)** *with the update Y(m)[i], denoted $X(n)[j] \vdash Y(m)[i]$, iff: i) $m < n$, and ii) X(n)[j] is received at the DW* **before** *the answer QR(m)[j] of update Y(m)[i] is received at DW.*

Notation	Meaning
IS[i]	Information source with subscript i.
X(n)[i]	X is update (SC or DU) from IS[i] at sequence number n.
Q(n)	Query used to handle update X(n)[i].
Q(n)[i]	Sub-query of Q(n) sent to IS[i].
QR(n)[i]	Query result of Q(n)[i].
QR(n)	Query result of Q(n) after re-assembly of all QR(n)[i] for all i.

Fig. 2. Notations and Their Meanings.

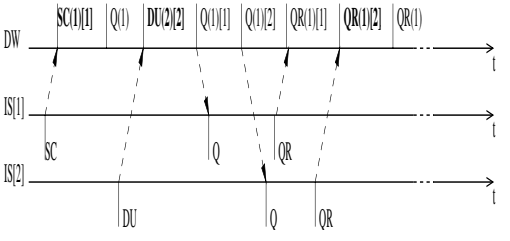


Fig. 3. Time Line for a Maintenance Concurrent Data Update.

Figure 3 illustrates the concept of a **maintenance-concurrent** update defined in Definition 1 with a time line. Messages only get time stamps assigned upon their arrival at the DW layer. That means that the **maintenance-concurrent** update is defined with respect to the data warehouse layer instead of the IS layer. Assume we have one DW and two IS[1] and IS[2]. First, there is a schema change SC at IS[1]. Then, there is a data update DU at IS[2]. From the figure, we can see that the SC is received by the DW before the DU, but DU occurs at IS[2] before the adaptation query Q(1)[2] of SC arrives at IS[2], that is, DU occurs before the query result QR(1)[2] arrives at the DW. So, here the DU is maintenance concurrent with SC by Definition 1.

There are four types of **maintenance-concurrent** updates listed in Table 1 in the order of the easiest to the hardest in terms of handling them.

Definition 2. *A query is called a* **broken query** *if it cannot be processed by an IS because the schema of the IS expected by the query is not consistent with the actual schema of the IS encountered during query processing.*

Type	Meaning A **maintenance-concurrent** DU happened	Denoted By
I	when handling DU	$DU_h - DU_{mac}$
II	when handling SC	$SC_h - DU_{mac}$
III	when handling DU	$DU_h - SC_{mac}$
IV	when handling SC	$SC_h - SC_{mac}$

DU_h (or SC_h) denotes the DU (or SC) that is currently being handling by the DyDa system. DU_{mac} (or SC_{mac}) denotes the DU (or SC) that is a **maintenance-concurrent** DU (or SC).

Table 1. Four Types of Maintenance-Concurrent Updates.

We distinguish between three types of broken queries based on the type of SC causing the problem as well as the characteristics of the IS space available for the system to deal with this problem.

- **Type 1: Broken Queries caused by a RenameSC.** In this case the data is still there but only the interface of this IS relation has changed. We can use a name mapping strategy to get to the desired data.
- **Type 2: Broken Query caused by a DropSC with Replacement.** While the data has removed from the IS, we are able to find the required information from an alternate source that holds duplicate data.
- **Type 3: Broken Query caused by DropSC without Replacement.** The data has really been dropped from the IS, and the system is not able to identify an alternate source for the data.

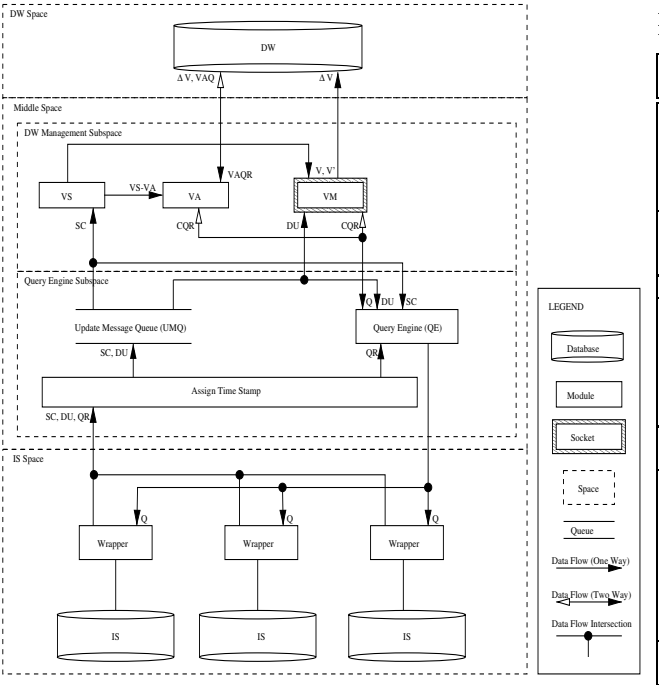
AddSC will not result in broken queries as they do not interfere with the interpretation of any existing query. A broken query as defined in Definition 2 will be returned by the respective IS as an empty query result with a "broken query" error message.

3 The DyDa Framework

3.1 Overall Solution Architecture

The DyDa framework is divided into three spaces: DW space, middle space, and IS space as described in Figure 4, which depicts the modules of our proposed framework and the data flow between them. The DW space houses the extent of the DW. It receives queries from the middle space bundled together with the data to update the DW. The IS space is composed of ISs and their corresponding wrappers. Wrappers are responsible for translating queries, returning query results, and sending notifications of DUs and SCs of the ISs to the middle space.

The middle space is the integrator of the DyDa framework. It can be divided into two subspaces. The higher-level subspace is called the **DW management subspace**. All the DW management algorithms, like VS, VA and VM, are located in this subspace. The lower-level subspace is called the **query engine**



Meaning of Symbols Used in Figure 4.

Symbols	Meaning
V	View definition affected by either Schema Change (SC) or Data Update (DU).
V'	Evolved view definition of affected view V.
DW	Data warehouse.
ΔV	Incremental view extent of data warehouse, i.e., set of tuples to be inserted into or removed from the extent of view V.
CQR	Query result that is returned by QE.
VS-VA	All information VA module requires from VS module for view adaptation. It includes: V, V', Meta-knowledge, Synchronization-mapping.
VAQ	View Adaptation Query.
VAQR	View Adaptation Query Result.

Fig. 4. Architecture of DyDa Framework

subspace. This subspace is composed of the Query Engine (QE) and its supporting modules, namely, the Update Message Queue (UMQ) module and Assign Time Stamp module. The UMQ module captures all updates received, and the Assign Time Stamp module is used to assign a time stamp to each incoming message, i.e., SC, DU and query result, based on their arrival. The two subspaces effectively correspond to two different levels of concurrency control. The key idea is that **maintenance-concurrent** DUs will be handled locally by the QE module at the lower-level of the middle space, so that DW management algorithms at the upper-level, such as VS, VM and VA, are shielded from and will never be aware of any **maintenance-concurrent** DUs. While our associated technical report [13] gives more details on the dynamicity and protocol of the system, we now discuss the justification of the two-layered concurrency control in Section 4.

4 Two Levels of Concurrency Control of DyDa

While the detection of concurrent DUs is based on the local timestamp assigned to DUs upon their arrival at the DW [12], the detection of concurrent SCs is based on identifying when and why a submitted query is broken and hence returned unanswered.

In the DyDa framework in order to separate out the handling of concurrent data updates and schema changes, there are two levels of concurrency control corresponding to the two sub-spaces of the *middle* space. At the lower level of concurrency control (the *query engine subspace*), the concurrent DUs as well as the concurrent SCs of the type **RenameSC** will be handled by the QE. The *DW management subspace* supports the higher level of concurrency control. The management algorithms (e.g., VM, VS and VA) at that subspace cooperate with each other in order to handle schema changes of the type **DropSC**. **AddSC** SCs do not render view definitions undefined and hence do not affect VM, VS nor VA. Thus they do not need to be handled by DyDa.

4.1 Low Level Concurrency Control at QE Subspace

DyDa solves the type *I* and *II* **maintenance-concurrent** problems (see Table 1) at the QE level by the local correction strategy and part of type *III* and *IV* **maintenance-concurrent** problems (see Table 1) by a local naming mapping strategy.

Using Local Correction to Handle Maintenance-Concurrent DUs. All queries from the DW modules down to the IS space will first go through the QE module (Figure 4). This includes incremental view maintenance queries, view adaptation queries, or view recomputation queries. Given that all three query types are "extent-related queries", the query engine will use the local correction (LC) algorithm described in [14] to successfully fix all side effects of concurrent DUs on these queries before passing the corresponding query results up to the next layer of the system. This results in the concurrent DUs to be transparent to all modules in *DW management* subspace. By using the local correction algorithm, there is no possibility to be faced with an infinite wait due to the recursive correction of queries, as is a problem in the Strobe solution [16]. Details of the local correction scheme we adopt for DyDa and its proof of correctness are beyond the scope (and space limitations) of this paper, but can be found in [12].

Using Name Mapping to Handle Maintenance-Concurrent RenameSCs In addition to the above service, features based on a temporary name mapping table have been added to the query engine of DyDa in order to handle concurrent SCs of type **RenameSC**. Whenever there is a query result received, the QE will check whether there is a concurrent renameSC received in the UMQ. If there is no concurrent renameSC, QE will directly return the result. If there is one concurrent renameSC that makes the query broken, QE will rename the query based on the renameSC and resent it again until succeed and keep the renaming transparent to the higher layer. More comprehensive discussion of handling renameSC at QE can be found in [13].

If more than one SC is found in the UMQ, and at least one of them is not RenameSC (i.e., at least one DropSC exists), then the QE cannot handle the

broken query anymore. It will report an error message to the *data warehouse maintenance* subspace. If the modified query based on the RenameSC breaks again due to another RenameSC in the UMQ, the QE module will modify the query and resubmit it until either the query succeeds, or a DropSC is encountered in the UMQ. In the highly unlikely case that the ISs were to continuously issue new RenameSCs, then the QE would have to endure a long processing time.

4.2 High Level Concurrency Control at DW Management Subspace

Finally, the VS, VM and VA modules in the *DW management subspace* finally need to handle the concurrent DropSC problem. VS has the least effect of the concurrent DropSCs, since it will not send queries to QE as shown below. VM module is affected by the concurrent DropSCs, but doesn't have the capability to handle them as shown below. VA module is the module designated to handling the concurrent DropSCs as shown in [13].

The VS Module and Maintenance-Concurrent SCs. From Definition 1, we know that VS will never have any trouble with concurrent SCs, because it will not send any queries down to the IS space. However, the VS module needs to provide information to the VA module to help VA to adapt the view extent under such concurrent SCs (Figure 4). From the state diagram (Figure 1) in Section 2, we can see that for every SC, the VS module will be called to generate a new version of the view definition to account for this SC.

From the point of view of VS, all SCs happen sequentially. If two SCs come to the *middle* space at the same time, they will be assigned a random handling order.[1] In a distributed environment as we are assuming, there is no convenient way to determine when the two SCs happened at the IS space relative to each other (unless they both come from the same IS). Plus they may indeed have truly occured at the same time. The issue of which of the SCs happened first for two different SCs from two autonomous ISs does not affect the correctness of the VS.

As further detailed in [13] after we apply *VS*, the *VA* module will know how each view definition evolved and thus is capable of generating the corresponding view adaptation queries for it. If *VS* drops a view as being no longer salvageable, i.e., empty extent, then *VA* doesn't need to adapt it.

The VM Module and Maintenance-Concurrent SCs. If VM encounters a concurrent SC, it recognizes that by the fact that it will receive a broken query passed up from the QE module. Because the VM algorithms in the literature haven't considered thus far how to handle broken queries, we put the responsibility of this handling on the VA module that needs to be redesigned anyway.

[1] If two SC changes come to the middle space of DyDa framework at same time, we will pick a preferred order of handling the SCs based on a quality-cost model described in [4]. The decision of the handling order of a set of SCs is out of the scope of this paper.

Thus in our system existing VM algorithms from the literature are kept intact. Hence the VM algorithm simply stops (abnormally) and sends the DU that it is currently handling to the VA module. That DU will then be handled later by the VA algorithm when it is adapting the extent of the view as discussed in detail in [13]. Given that we isolate the VM module from the handling of **maintenance-concurrent** SCs, all the VM algorithms, e.g., PSWEEP [12], SWEEP [1], Strobe [16], and ECA [15] could be used in DyDa.

4.3 Effect of Maintenance-Concurrent Updates on Existing DW Management Algorithms

In the DyDa framework, we decide to let the query engine (QE) module as explained in Section 3 fix the problem of any **maintenance-concurrent** DU before the query results reach the DW management modules. So, the **maintenance-concurrent DUs** have no effect on the three modules VA, VM and VS. However, the three modules have a different degree of awareness of **maintenance-concurrent SCs**, namely:

- There is no concept of **maintenance-concurrent** SCs for the VS module, because the VS module never sends any query down to the IS space.
- While the VM module will send queries down to the IS space for view maintenance, it assumes the view definition that it is maintaining will not be changed. So if a maintenance query is broken by a **maintenance-concurrent** SC, the VM module has to be reset by the DyDa system so to work with the newly updated view definition that has been generated by VS to take care of that **maintenance-concurrent** SC. VM itself however can function unchanged in our environment.
- The VA module also sends down queries to the IS space to adapt the view extent. If a view adaptation query is broken by a **maintenance-concurrent** SC, then VA needs to handle this in its adaptation process. Hence, a completely new VA algorithm has been designed for DyDa [13].

5 Conclusions

In this paper, we first identify the broken query problem of DW management under concurrent DUs and SCs. Then, we propose the DyDa solution framework that solves the problem using a two layered architecture as presented in the paper. To our knowledge, our work is the first to address the data warehouse maintenance problem under fully concurrent DUs and SCs of ISs. DyDa overcomes the limitation of the only previous approach by dropping its restrictive assumption of cooperation of the ISs [14] We are in the process of implementing this DyDa data warehouse solution In the future, we plan to run experimental studies to assess performance aspects of this solution.

Acknowledgments. We would like to thank all DSRG members, in particular, A. Koeller and Y. Li for helping to implement the EVE system.

References

[1] D. Agrawal, A. E. Abbadi, A. Singh, and T. Yurek. Efficient View Maintenance at Data Warehouses. In *Proceedings of SIGMOD*, pages 417–427, 1997.

[2] A. Gupta, I. Mumick, and K. Ross. Adapting Materialized Views after Redefinition. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 211–222, 1995.

[3] L. V. S. Lakshmanan, F. Sadri, and I. N. Subramanian. SchemaSQL — A Language for Interoperability in Relational Multi-database Systems. In T. M. Vijayaraman et al., editors, *Proceedings of the twenty-second international Conference on Very Large Data Bases*, pages 239–250, Mumbai, India, Sept. 1996.

[4] A. J. Lee, A. Koeller, A. Nica, and E. A. Rundensteiner. Non-Equivalent Query Rewritings. In *International Database Conference*, Hong Kong, July 1999.

[5] S. Marche. Measuring the Stability of Data Models. *European Journal of Information Systems*, 2(1):37–47, 1993.

[6] M. Mohania and G. Dong. Algorithms for Adapting Materialized Views in Data Warehouses. *International Symposium on Cooperative Database Systems for Advanced Applications*, December 1996.

[7] A. Nica and E. A. Rundensteiner. View Maintenance after View Synchronization. In *International Database Engineering and Applications Symposium (IDEAS'99)*, August, Montreal, Canada 1999.

[8] D. Quass and J. Widom. On-Line Warehouse View Maintenance. In *Proceedings of SIGMOD*, pages 393–400, 1997.

[9] E. A. Rundensteiner, A. Koeller, and X. Zhang. Maintaining data warehouses over changing information sources. *Communications of the ACM*, June 2000.

[10] E. A. Rundensteiner, A. Koeller, X. Zhang, A. Lee, A. Nica, A. VanWyk, and Y. Li. Evolvable View Environment. In *Proceedings of SIGMOD'99 Demo Session*, pages 553–555, May/June 1999.

[11] D. Sjoberg. Quantifying Schema Evolution. *Information and Software Technology*, 35(1):35–54, January 1993.

[12] X. Zhang, L. Ding, and E. A. Rundensteiner. PSWEEP: Parallel View Maintenance Under Concurrent Data Updates of Distributed Sources. Technical Report WPI-CS-TR-99-14, Worcester Polytechnic Institute, Dept. of Computer Science, May 1999.

[13] X. Zhang and E. A. Rundensteiner. DyDa: Dynamic Data Warehouse Maintenance in a Fully Concurrent Environment. Technical Report WPI-CS-TR-99-20, Worcester Polytechnic Institute, Dept. of Computer Science, July 1999.

[14] X. Zhang and E. A. Rundensteiner. The SDCC Framework for Integrating Existing Algorithms for Diverse Data Warehouse Maintenance Tasks. In *International Database Engineering and Application Symposium*, pages 206–214, Montreal, Canada, August, 1999.

[15] Y. Zhuge, H. García-Molina, J. Hammer, and J. Widom. View Maintenance in a Warehousing Environment. In *Proceedings of SIGMOD*, pages 316–327, May 1995.

[16] Y. Zhuge, H. García-Molina, and J. L. Wiener. The Strobe Algorithms for Multi-Source Warehouse Consistency. In *International Conference on Parallel and Distributed Information Systems*, pages 146–157, December 1996.

Scalable Maintenance of Multiple Interrelated Data Warehousing Systems*

Lingli Ding, Xin Zhang and Elke A. Rundensteiner

Department of Computer Science
Worcester Polytechnic Institute
Worcester, MA 01609-2280
(lingli| xin | rundenst)@cs.wpi.edu

Abstract. The maintenance of data warehouses(DWs) is becoming an increasingly important topic due to the growing use, derivation and integration of digital information. Most previous work has dealt with one centralized data warehouse only. In this paper, we now focus on environments with multiple DWs that are possibly derived from other DWs. In such a large-scale environment, data updates from base sources may arrive in individual data warehouses in different orders, thus resulting in inconsistent data warehouse extents. We propose to address this problem by employing a registry agent responsible for establishing one unique order for the propagation of updates from the base sources to the DWs. With this solution, individual DW managers can still maintain their respective extents autonomously and independently from each other, thus allowing them to apply any existing incremental maintenance algorithm from the literature. We demonstrate that this **r**egistr**y**-based **co**ordination approach (RyCo) indeed achieves consistency across all DWs.

Keywords: Distributed Data Warehousing, View Maintenance, Registry.

1 Introduction

Data warehousing [14, 6, 7, 15, 11, 10] (DW) is a popular technology to integrate data from heterogeneous information sources (ISs) in order to provide data to, for example, decision support or data mining applications [2]. Once a DW is established, the problem of maintaining it consistent with underlying ISs under updates remains a critical issue. It is popular to maintain the DW incrementally [1, 9, 8] instead of recomputing the whole extent of the DW after each IS update, due to the large size of DWs and the enormous overhead associated with the DW

* This work was supported in part by several grants from NSF, namely, the NSF NYI grant #IRI 97–96264, the NSF CISE Instrumentation grant #IRIS 97–29878, and the NSF grant #IIS 97–32897. Dr. Rundensteiner would like to thank our industrial sponsors, in particular, IBM for the IBM partnership award, and GTE for partial support of Xin Zhang.

Y. Kambayashi, M. Mohania, and A M. Tjoa (Eds.): DaWaK 2000, LNCS 1874, pp. 104–113, 2000.
© Springer-Verlag Berlin Heidelberg 2000

loading process. The majority of such view maintenance algorithms [1, 17, 16] as of now are based on a centralized DW system in which materialized views are stored in a single site even if bases may be distributed.

Given the growth of digital on-line information and the distributed nature of sources found on the Internet, we can expect large collections of interrelated derived repositories instead of just one individual DW. Such a distributed environment composed of data warehouses defined on top of each other is also emerging as a design solution for a growing set of applications [13]. In this paper, we focus on the view maintenance in such distributed environments.

1.1 Related Work

Zhuge et al. [18] proposed an architecture for handling multiple views consistency for several views specified in a single data warehouse. Stanoi et al. [12] proposed a weak consistency maintenance approach that processes the cumulative effect of a batch of updates. In their latter work [13], they proposed an incremental algorithm for maintenance distributed multi-view DWs. Their approach requires storage for a stable state with table of changes. A stable state is a snapshot of a *safe* state in the view's history such that the view will not need to answer update queries based on a state prior to this safe state. The table of changes includes all the updates following the safe state of the actual materialized view. There is a dependency list appended to all entries in the table of changes [13]. A materialized view is refreshed only when the new state is considered safe, i.e., when the respective updates are in sequence, and have been integrated by all the corresponding direct descendents. Hence, DWs always need to wait for their corresponding direct descendents. In our approach, we don't need any DW to keep a safe state table. In fact, our solution is able to eliminate the possibility of an infinity wait. Our approach has good performance for both a flat or a tall topology of DW dependencies and hence is scalable.

1.2 Our RyCo Approach

In this paper, we propose a registry-based coordination solution strategy that is able to coordinate the maintenance of multiple data warehouses in a non-intrusive manner by establishing a unique order among all base update notifications for the environment. DW managers exploit this order synchronization to determine how and when to process incoming update notifications and to commit their respective extent updates, instead of blindly making use of the order in which the update messages are received from the other DWs. We demonstrate that this method, which we call RyCo for registry-based coordination for distributed data warehouse maintenance, will indeed achieve consistency across all data warehouses with little overhead. In summary, RyCo has the following advantages:

- First, each DW is maintained by a separate DW manager and can be updated independently from other DWs.

- Second, any DW maintenance algorithm for a central DW can be adopted from the literature and plugged into our system.
- Third, all DWs are guaranteed to be maintained consistent with each other and the ISs at all times.
- Fourth, unlike previous work [13], our approach avoids any requirement for a safe time before one can refresh a DW.

Outline. The rest of the paper is organized as follows. The system definitions, assumptions and view consistency levels are presented in Section 2. Section 3 presents the architecture and algorithm of the registry-based solution. Finally, we conclude in Section 4.

2 Distributed Data Warehousing Environments

2.1 View Dependency Graph

A distributed data warehousing environment (DDWE) is composed of multiple possibly interrelated DWs. Individual DWs are independent running possibly on different sites. Each DW has its own views. Views may be defined on views of its own DW, on other views, on base relations in ISs, or on a combination of the above. For simplicity, we assume each DW only has one view and each IS has one base relation. We can use the MRE wrapper proposed in [4] to release the later assumption.

Definition 1. *A view V is defined as $V = S_1 \bowtie S_2 \bowtie ... \bowtie S_n$ where S_i ($1 \leq i \leq n$) could be either a base relation or a view. We say S_i is a* **parent** *of the view V and V is the* **direct descendent** *of S_i. The bases B_1, ..., B_m from which V is ultimately derived, i.e., all ancestors of V that are bases, are called* **base-ancestor**s *of V. If a base B is a base-ancestor of both a view V_1 and a view V_2, then we say B is a* **common base-ancestor** *of V_1 and V_2.*

A view-dependency graph [3] represents the hierarchical relationships among the views in DWs and bases in ISs. The dependency graph shows how a view is derived from bases and/or other views.

Definition 2. *The* **view dependency graph** *of the view V in a DDWE is a graph $G_v = (N, E)$ with N the set of bases or views that are ancestors of V including V itself and E the set of of directed edges $E_{ij} = (N_i, N_j)$ for each N_j that is a direct descendent of N_i, where $N_i, N_j \in N$. All nodes N_i are labeled by the name of the bases or views that they represent.*

Example 1. Suppose there are two base relations B_1, B_2 in two ISs and three views V_0, V_1 and V_2 at three DWs that reside on different sites, defined as $V_0 = V_1 \bowtie V_2$, $V_1 = B_1 \bowtie B_2$ and $V_2 = B_1 \bowtie B_2$.

Figure 1 depicts the view dependency graph for the views. For example, views V_1 and V_2 are **parents** of the view V_0. B_1 and B_2 are **ancestors** of V_0. V_0 is the **direct descendent** of V_1 and V_2. The **base-ancestors** of V_0 (as well as V_1) are B_1 and B_2; while B_1 and B_2 are also **common-bases ancestors** of V_1 and V_2.

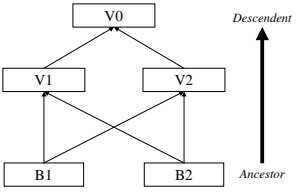


Fig. 1. A Distributed DW System Example

2.2 Base Consistency and View Consistency

Consider a base B that executes a sequence of atomic data updates. These changes result in a sequence of states. We use the notation $St_m(B)$ for the m'th state in this sequence.

Definition 3. *Let a view V be defined by Definition 1. V is said to be* **base consistent** *if each DW state of V, $St_i(V)$, corresponds to one single state $St_j(B_k)$ for each base B_k with k from 1 to m.*

Based on Definition 3, a view is base consistent if at all times its extent reflects at most one real state of each IS. V cannot reflect two different states of a base B_j.

Definition 4. *Two views V_1 and V_2 are* **view consistent** *with each other if V_1 and V_2 are both base consistent and after committing each data update (ΔB) from any of the common bases B at their respective sites, states $St_i(V_1)$ and $St_j(V_2)$ correspond to the same state $St_m(B)$ of that common base B.*

2.3 View Maintenance in Distributed Data Warehouse Systems

To distinguish view updates caused by different data updates, we use a data update identifier in the rest of the paper, in short DU-id, to identify DUs from different ISs. For simplicity, we use an integer to denote DU-id with the first digit of the DU-id the same as the IS index number [1] and rest of DU-id is the index of DUs from that IS. For example, $\Delta S_i/kj$ denotes the ΔS_i calculated for the jth update from B_k.

Assume the view V is defined as in Definition 1 and the views S_i (for $1 \leq i \leq n$) have multiple common bases B_k. If there is any base data update from such a common base B_k, denoted as $\Delta B_k/kj$, the views S_1, ..., S_n and V need to be maintained **view consistent** with each other. Then all $\Delta S_i/kj$ (i from 1 to n) are calculated where some of the $\Delta S_i/kj$ could be null if B_k is not a base of the view S_i, and sent to the view V. If all $\Delta S_i/kj$ only contain the effect of the data update of $\Delta B_k/kj$ for the base B_k, we can calculate the new view extent V as follows:

[1] If there were a lot of bases, we could use more digits to represent base index number.

$$V + \Delta V = (S_1 + \Delta S_1/kj) \bowtie ... \bowtie ... \bowtie (S_i + \Delta S_i/kj) \bowtie ... \bowtie (S_n + \Delta S_n/kj) \quad (1)$$

Hence, the ΔV expression is shown as follows:

$$\begin{aligned} \Delta V &= \Delta S_1/kj \bowtie S_2 \bowtie ... \bowtie S_i \bowtie ... \bowtie S_n \\ &+ (S_1 + \Delta S_1/kj) \bowtie \Delta S_2/kj \bowtie S_3 \bowtie ..... \bowtie S_i \bowtie ... \bowtie S_n + ... \\ &+ (S_1 + \Delta S_1/kj) \bowtie ... \bowtie (S_{i-1} + \Delta S_{i-1}/kj) \bowtie \Delta S_i/kj \bowtie S_{i+1} \bowtie ... \bowtie S_n + ... \\ &+ (S_1 + \Delta S_1/kj) \bowtie ... \bowtie (S_i + \Delta S_i/kj) \bowtie ... \bowtie (S_{n-1} + \Delta S_{n-1}/kj) \bowtie \Delta S_n/kj \end{aligned} \quad (2)$$

However, we note that multiple data updates may have happened concurrently at different bases. These DUs may arrive at different views in a different order. Some of the $\Delta S_i/kj$ may not only contain the effect of $\Delta B_k/kj$ but in addition they may also incorporate the effect of some other DUs. In other words, S_i and S_j $(i \neq j)$ are not guaranteed to be view consistent with each other. Hence, view V derived from S_i $(1 \leq i \leq n)$ is not base consistent.

To maintain view V base consistent by Definition 4, we need to update the view V based on the update messages from all its parents S_1, S_2, ... S_n that reflect the same state of the same base. That is all the $\Delta S_i/kj$ (i from 1 to n) must reflect the same states of all the bases B_1, ..., B_m. We can't control that S_i and S_j (i ≠ j) always reflect the same base extents because S_i and S_j belong to different DWs. But to maintain a view V correctly, we first need to assure all its parents S_i to be view consistent with each other.

3 The Registry-Based Coordination for Distributed DW Maintenance

3.1 View Maintenance Order and Consistency

If DWs are maintained independently from other DWs, then they may not be consistent with each other. It is difficult to maintain a view that is defined on top of already inconsistent views to be base consistent.

Definition 5. *The order in which a DW receives data updates is called* **receive-message-order**, *and the order in which a DW updates its extent* **update-message-order**.

Lemma 1. *Given a set of views V_i that are view consistent with each other. Then any descendent view V defined directly on top of these consistent views V_i can be maintained as base consistent and view consistent with V_i.*

DWs could have a different receive-message-order (**RMO**) and no knowledge of the receive-message-order of other DWs. If DWs were to be maintained not based on their respective receive-message-order but rather on the same enforced

update-message-order(**UMO**), then the DWs would be updated and would reflect base changes in the same order. Hence DWs are consistent with each other and thus are base consistent.

To generate such a unique update-message-order, the key idea is to use the registry agent. All DWs and bases are registered with the registry. The registry generates a unique update-message-order and sends it to all DWs in the system. All the DW extents are updated by this update-message-order and hence all DWs are then consistent with each other.

3.2 System Architecture of RyCo Approach

A **r**egistr**y**-based **co**ordination (RyCo) framework for distributed DW maintenance is composed of three types of components as depicted in Figure 2. First, the registry agent is used to generate a unique update order. Second, one wrapper at each base sends DU-ids and data updates to related views. It also processes queries sent by direct descendent. Third, the mediator of each data warehouse maintains the data warehouse based on the updates in the order decided by the registry. Figure 2 depicts the relationships between these three components. As we can see there is one single registry that connects to multiple base wrappers and mediators of different DWs.

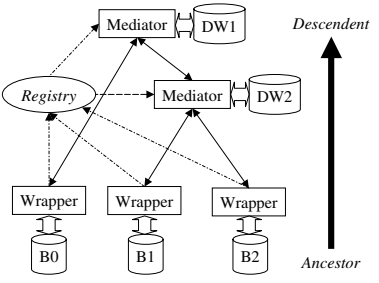


Fig. 2. Overall Architecture of RyCo System

Figure 3 shows the simple structure of the registry. The registry keeps the information of registered DWs in the DW list. It can further add or remove a DW from its DW list when a DW is added or removed from DDWE. It receives DU-id from all bases and orders those IDs based on their receive order and then forwards those IDs in that unique order to all DWs in the system. The bases as shown in Figure 4 only send a DU-id to the registry, as there is no need to send real update messages. The base will still send data update messages with DU-ids to all their direct descendent DWs as before and also answer all possible queries from its direct descendent.

In Figure 5, a DW mediator consists of the VM (View Manager) and the Query Processor. The VM has the same function as in centralized DW sys-

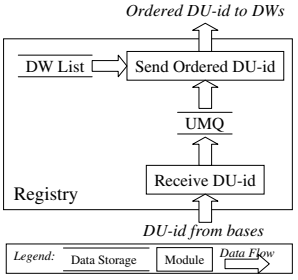


Fig. 3. Structure of Registry

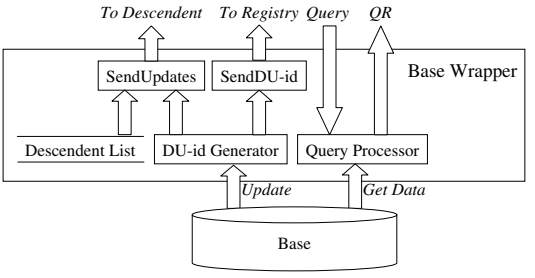


Fig. 4. Structure of Base Wrapper

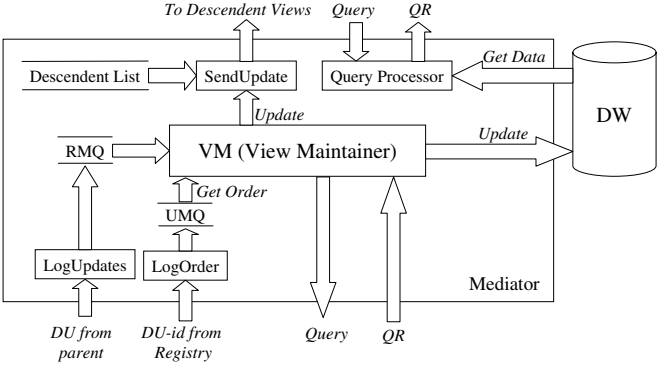


Fig. 5. Structure of DW Mediator

tems. The VM processes one update at a time and updates the DW view to be consistent with the base after this update. Any incremental view maintenance algorithm from the literature, e.g., SWEEP [1] could be employed as VM here with minor modification to that the new extent is computed based on the order from the update-message-order, and not the receive-message-order when the updates are actually propagated upward through the dependency graph.

A DW mediator in the system has two queues, namely the receive-message-queue (RMQ) and update-message-queue (UMQ). The RMQ buffers update messages received from its parents in the receive-message-order. The UMQ is responsible for buffering ordered DU-id messages from the registry. Views are maintained and updated in the order of the UMQ as follows. First, VM removes the next DU-id from the head of the UMQ and checks whether the DU-id is related to this view. Because the registry will send all DU-ids to all the views, some of them may not be related to this view. If the DU-id is not related, VM will send a empty update [2] with the same DU-id to all direct descendent DWs. Otherwise, VM will wait for all updates with this DU-id from all its parents,

[2] That means no update for this specific base DU.

and then incrementally calculate ΔV for those data updates using the Equation 2. Then, the VM updates its local DW with ΔV. As last, the mediator sends the update with the same DU-id to all direct descendent DWs.

The Query Processor component is used to process any query sent from its children. Its functionality is similar to an IS query processor in centralized DW systems.

We can observe that, using the registry, all DWs in our distributed data warehousing environment incorporate the DUs in the same order. Hence they are consistent with each other and all views are base consistent. More details on the RyCo solution, including formal theorems and some cost evaluation, can be found in our technical report [5].

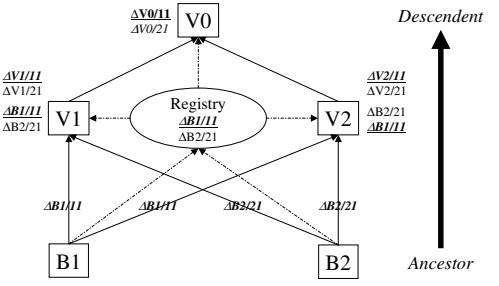


Fig. 6. RyCo View Maintenance Example

3.3 Illustrating Example of RyCo Process

The example shows how the registry is used to enable view maintenance in distributed data warehousing environments. The pseudo-code of the RyCo algorithm is given in the technical report [5].

Example 2. We use the view definition in the example 1. Assume there is a data update at B_1 denoted by $\Delta B_1/11$ and a data update at B_2 denoted by $\Delta B_2/21$. These two DUs are then sent to views V_1 and V_2. Assume that data updates $\Delta B_1/11$ and $\Delta B_2/21$ arrive at the views V_1 and V_2 in different orders.

Without registry, view maintenance for V_1 computes and updates V_1 with $\Delta V_1/11$ for $\Delta B_1/11$ and then $\Delta V_1/21$ for $\Delta B_2/21$ based on the RMO. However, the view manager of V_2 computes and updates V_2 in a reverse order, i.e., $\Delta V_2/21$ for $\Delta B_2/21$ first and then $\Delta V_2/11$ for $\Delta B_1/11$. V_1 sends its update messages $\Delta V_1/11$, $\Delta V_1/21$ and V_2 sends $\Delta V_2/21$ and $\Delta V_2/11$ to V_0. We notice that $\Delta V_1/11$ reflects the new state of B_1 but not B_2. But $\Delta V_2/11$ reflects both the new state of B_1 and B_2. When V_0 computes its new extent to reflect the update from base B_1 based on the update messages $\Delta V_1/11$ and $\Delta V_2/11$, its new extent

incorporates aspects of two different states of B_2, i.e., the B_2 before and after update $\Delta B_2/21$. Hence, V_0 is inconsistent with base.

With our registry agent, we can solve this problem easily. Assume that the registry receives DU-ids in the order of 11 and 21. Then the registry forwards this unique update-message-order to all views, namely V_1, V_2 and V_0. The new view extents of V_1 and V_2 are calculated and updated in the update-message-order. Then $\Delta V_1/11$, $\Delta V_1/21$, $\Delta V_2/11$, and $\Delta V_2/21$ are sent to the view V_0 in the order of 11, 21. $\Delta V_1/11$ and $\Delta V_2/11$ only have the effect of the data update $\Delta B_1/11$. $\Delta V_1/21$ and $\Delta V_2/21$ have the effect of both $\Delta B_1/11$ and $\Delta B_2/21$. The view V_0 is then updated in the same update-message-order. That is, it first incorporates the effect of $\Delta B_1/11$ based on $\Delta V_1/11$ and $\Delta V_2/11$, then incorporates the effect of $\Delta B_2/21$ based on $\Delta V_2/21$, $\Delta V_2/21$. Because views V_1, V_2 and V_0 are updated by the same update-message-order generated by the registry, the views are consistent with each other and thus also are base consistent.

4 Conclusion

In this paper, we propose the RyCo approach for consistent view maintenance in distributed data warehouse environment. All the views in our system are maintained and updated independently according to the notification order from the registry. Our registry-based solution does not need any safe time to refresh a materialized view, as required by the only alternate solution in the literature thus far [13].

In the future, to make the system more scalable in terms of the number of bases and views in the system, we plan to optimize RyCo by dividing the system into different clustered groups, called DW groups (DWG) [5]. Each group is equipped with its own dedicated registry. Such a distributed registry approach could still keep the DWs consistent with each other while we reduce the unrelated and empty messages sent by the registry and handled in each DW group.

References

[1] D. Agrawal, A. E. Abbadi, A. Singh, and T. Yurek. Efficient View Maintenance at Data Warehouses. In *Proceedings of SIGMOD*, pages 417–427, 1997.

[2] S. Chaudhuri and U. Dayal. An Overview of Data Warehousing and OLAP Technology. *SIGMOD Record*, 26(1):65–74, 1997.

[3] L. Colby, A. Kawaguchi, D. Lieuwen, I. Mumick, and K. Ross. Supporting Multiple View Maintenance Policies. *AT&T Technical Memo*, 1996.

[4] L. Ding, X. Zhang, and E. A. Rundensteiner. The MRE Wrapper Approach : Enabling Incremental View Maintenance of Data Warehouses Defined On Multi-Relation Information Sources. In *Proceedings of the ACM First International Workshop on Data Warehousing and OLAP (DOLAP'99)*, pages 30–35, November 1999.

[5] L. Ding, X. Zhang, and E. A. Rundensteiner. Scalable Maintenance of Multiple Interrelated Data Warehousing Systems. Technical Report WPI-CS-TR-00-16, Worcester Polytechnic Institute, Dept. of Computer Science, May 2000.

[6] H. García-Molina, W. Labio, J. L. Wiener, and Y. Zhuge. Distributed and Parallel Computing Issues in Data Warehousing . In *Symposium on Principles of Distributed Computing*, page 7, 1998. Abstract.

[7] A. Gupta and I. S. Mumick. What is the data warehousing problem? (Are materialized views the answer?). In *International Conference on Very Large Data Bases*, page 602, 1996. Panel.

[8] A. Kawaguchi, D. F. Lieuwen, I. S. Mumick, and K. A. Ross. Implementing Incremental View Maintenance in Nested Data Models. In *Workshop on Database Programming Languages*, pages 202–221, 1997.

[9] M. K. Mohania, S. Konomi, and Y. Kambayashi. Incremental Maintenance of Materialized Views. In *Database and Expert Systems Applications (DEXA)*, pages 551–560, 1997.

[10] E. A. Rundensteiner, A. Koeller, and X. Zhang. Maintaining data warehouses over changing information sources. *Communications of the ACM*, June 2000.

[11] E. A. Rundensteiner, A. Koeller, X. Zhang, A. Lee, A. Nica, A. VanWyk, and Y. Li. Evolvable View Environment. In *Proceedings of SIGMOD'99 Demo Session*, pages 553–555, May/June 1999.

[12] I. Stanoi, D.Agrawal, and A. E. Abbadi. Weak Consistency in Distributed Data Warehouses. In *Proceedings of the International Conference of Foundations of Data Organization*, November 1998.

[13] I. Stanoi, D.Agrawal, and A. E. Abbadi. Modeling and Maintaining Multi-View Data Warehouses. In *Proceedings of the 18th International Conference on Conceptual Modeling (ER'99)*, pages 161–175, November 1999.

[14] M. Wu and A. P. Buchman. Research Issues in Data Warehousing. In *Datenbanksysteme in Büro, Technik und Wissenschaft*, pages 61–82, 1997.

[15] X. Zhang and E. A. Rundensteiner. The SDCC Framework for Integrating Existing Algorithms for Diverse Data Warehouse Maintenance Tasks. In *International Database Engineering and Application Symposium*, pages 206–214, Montreal, Canada, August, 1999.

[16] Y. Zhuge, H. García-Molina, J. Hammer, and J. Widom. View Maintenance in a Warehousing Environment. In *Proceedings of SIGMOD*, pages 316–327, May 1995.

[17] Y. Zhuge, H. García-Molina, and J. L. Wiener. The Strobe Algorithms for Multi-Source Warehouse Consistency. In *International Conference on Parallel and Distributed Information Systems*, pages 146–157, December 1996.

[18] Y. Zhuge, J. L. Wiener, and H. García-Molina. Multiple View Consistency for Data Warehousing. In *Proceedings of IEEE International Conference on Data Engineering*, pages 289–300, 1997.

View Maintenance for Hierarchical Semistructured Data *

Hartmut Liefke and **Susan B. Davidson**

Dept. of Computer and Information Science
University of Pennsylvania
liefke@seas.upenn.edu and susan@cis.upenn.edu

Abstract. The problem of incremental view maintenance in materialized data warehouses has been studied extensively for relational select-project-join (SPJ) views. Many new data sources, however, are highly irregular and views often perform complex restructuring operations. This paper describes WHAX (Warehouse Architecture for XML), an architecture for defining and maintaining views over hierarchical semistructured data sources with key constraints. The WHAX model is a variant of the deterministic model [5], but is more reminiscent of XML. The view definition language is a variation of XML-QL and supports selections, joins, and important restructuring operations such as regrouping and aggregation. The incremental maintenance is based on the notion of *multi-linearity* and generalizes well-known techniques from SPJ-views.

1 Introduction

XML has become an important standard for the representation and exchange of data over the Internet. As an instance of semi-structured data [1], which was initially proposed as a methodology in the Tsimmis project [7], it can also be used for data integration [8]. That is, data sources are represented in XML; transformations and integration are then expressed in one of several XML query languages that have been proposed (e.g. [12, 9, 8]) to create an XML view.

Once the view has been materialized, it must be maintained as updates to the underlying data sources are made. This problem has been studied for SPJ-views in the relational model (see [15] for a survey) and for object-oriented databases [16, 13]. SPJ-views have an important property: They are distributive with respect to relational union. For example, $R_1 \bowtie (R_2 \cup \triangle R_2) = (R_1 \bowtie R_2) \cup (R_1 \bowtie \triangle R_2)$ holds for any relations R_1 and R_2 and some new tuples $\triangle R_2$. To compute the view $R_1 \bowtie (R_2 \cup \triangle R_2)$, the result of $R_1 \bowtie \triangle R_2$ must be added to the existing view $R_1 \bowtie R_2$. This is ususally much more efficient than reevaluating the view.

More general, SPJ views are functions $f(R_1, ..., R_n)$ of (not necessarily distinct) base relations R_1, ..., R_n that are *multi-linear* with respect to union:[1] $f(R_1, ..., R_i \cup \triangle R_i, ..., R_n) = f(R_1, ..., R_i, ..., R_n) \cup f(R_1, ..., \triangle R_i, ..., R_n)$. Multi-linearity is a powerful concept. It allows efficient view maintenance for bulk updates and is also fundamental for other optimization techniques, such as parallel query evaluation and pipelining.

* This research was supported in part by DOE DE-FG02-94-ER-61923 Sub 1, NSF DBI99-75206, NSF IIS98-17444, ARO DAAG55-98-1-0331, and a grant from SmithKline Beecham.

[1] We slightly misuse the notion of multi-linearity: To be multi-linear, $\cup$ must be a group operation with an inverse function similar to the definition in [13].

Y. Kambayashi, M. Mohania, and A M. Tjoa (Eds.): DaWaK 2000, LNCS 1874, pp. 114–123, 2000.
© Springer-Verlag Berlin Heidelberg 2000

```
<Conf name="STACS" year="1996">
   <Publ> <Title> Views </Title>
          <Author> Tim </Author>
          <Author> Peter </Author>
          <Pages> <From> 117 </From>
                  <To> 127 </To>
          </Pages>
   </Publ>
   <Publ> <Title> Types </Title>
          <Author> Tim </Author>
          <Pages> <From> 134 </From>
                  <To> 146 </To>
          </Pages>
  </Publ>
</Conf>

<Person>
   <Name> Tim </Name>
   <Age> 35 </Age>
</Person>

<Person>
    <Name> Peter </Name>
    <Age> 45 </Age>
</Person>
```

Fig. 1. A Publication Database in XML

Motivating Example. Fig. 1 shows an XML document with conference publications. Each publication has a title, a list of author names and belongs to a conference. Each author name refers to a person.

In the semistructured model, nodes are identified by object identities and the update types are *edge insertion*, *edge deletion*, and *value modification* [2, 19]. Updates based on object identities, however, make it difficult to reason about what parts of the view have been affected.

To illustrate, consider the following simple view: store all authors who published in STACS'96. Furthermore, consider the insertion of a new author into some publication with OID p. Since it is not directly observable whether p belongs to STACS'96 or not, it is not clear whether the author should be inserted into the view. Existing algorithms [2, 19] therefore require auxiliary data structures to represent the relationship between view objects and those in the source. These complex data structures can be large and expensive to maintain. Another drawback of these approaches is that each atomic update causes queries to the source to update the view, which is inefficient for large update sequences.

The WHAX *Approach.* The Warehouse Architecture for XML (WHAX) is a new architecture that combines the power and simplicity of multi-linearity with the flexibility of the semistructured data model. The WHAX model is similar to the null-terminated deterministic model [5] and is also closely related to LDAP [18]. WHAX is based on the concept of *local keys*: Each outgoing edge of the same parent node is identified by some locally unique key. Hence, each node in the tree is uniquely identified by the path from the root.

Contributions. The query language WHAX-QL, based on XML-QL [12] generalizes SPJ-queries and allows powerful restructuring through regrouping and aggregations. We present a restriction of WHAX-QL that is provably multilinear, and hence allows efficient incremental maintenance. For deletion updates, we develop an extension of the counting technique used for SPJ-views. This technique can also be used for view definitions involving aggregation.

Limitations. In contrast to XML, the WHAX data model is *unordered*. Initial ideas for view maintenance of ordered data are briefly described in Sec. 7.

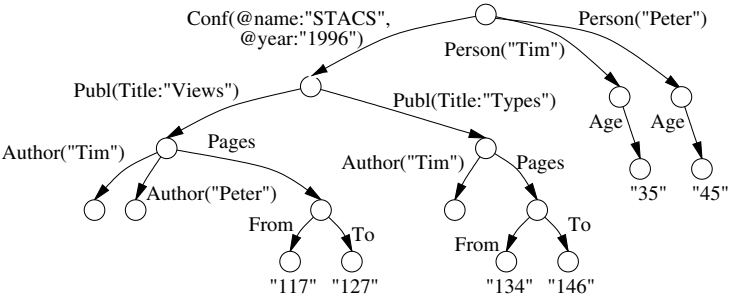


Fig. 2. Sample WHAX Tree

Furthermore, the WHAX model is tree-based and does not support navigation through graph structures as in the semistructured data model.

The rest of the paper is organized as follows. Sec. 2 describes the WHAX data model, followed by the view definition language in Sec. 3. We describe the multilinearity property for WHAX-views in Sec. 4 and extend the incremental view maintenance technique to deletions in Sec. 5. Sec. 6 describes how aggregate queries can be efficiently maintained in WHAX. We conclude in Sec. 7.

2 The WHAX Data Model

A WHAX data value is an unordered edge-labeled tree. The tree is *deterministic* in that the outgoing edges of a node have different *local identifiers*. Let B be the set of all base values (strings, integers, ...) and V be the set of WHAX-values. Local identifiers are pairs $l(k)$ with *label* $l \in B$ and *key* $k \in V$. The set of all WHAX-values is defined recursively as the (minimal) set of finite partial functions from local identifiers to trees: $V = (B \times V) \rightsquigarrow V$.

Values are constructed using $\{l_1(k_1) : \{...\}, ..., l_n(k_n) : \{...\}\}$. If k_i in $l_i(k_i)$ is the empty partial function $\{\}$, then we write l_i. If k_i is a tree construct $\{...\}$, then we write $l_i(...)$ instead of $l_i(\{...\})$. For example, the value $\{\mathsf{Conf}(@\mathsf{name} : \{\mathsf{STACS} : \{\}\}, @\mathsf{year} : \{1996 : \{\}\}) : ...\}$ represents conference STACS'96. Singleton trees of the form $\{\mathsf{str} : \{\}\}$ occur frequently and are abbreviated as double-quoted literals "str". Hence, the value above is equivalent to $\{\mathsf{Conf}(@\mathsf{name} : \text{“}\mathsf{STACS}\text{”}, @\mathsf{year} : \text{“}1996\text{”}) : ...\}$.

Fig. 2 shows the WHAX-value for the data in Fig. 1. We adopt the XPath[10] convention and preceed attribute labels with '@'. The Age label forms an identifier with empty key $\{\}$. Recall that "str" represents singleton function $\{\mathsf{str} : \{\}\}$.

Deep Union. The *deep union* operation [5] of two WHAX-trees v_1 and v_2 matches the common identifiers of v_1 and v_2 and recursively performs deep union on their respective subvalues. The edges that only occur in v_1 (or v_2), but not in v_2 (v_1, respectively) are copied into the resulting tree:[2]

$$\begin{aligned} v_1 \uplus v_2 ::= &\{l(k) : \ s_1 \uplus s_2 \mid l(k) : s_1 \in v_1, l(k) : s_2 \in v_2\} \cup \\ &\{l(k) : s \mid l(k) : s \in v_1, l(k) \notin \mathsf{dom}(v_2)\} \cup \\ &\{l(k) : s \mid l(k) : s \in v_2, l(k) \notin \mathsf{dom}(v_1)\} \end{aligned}$$

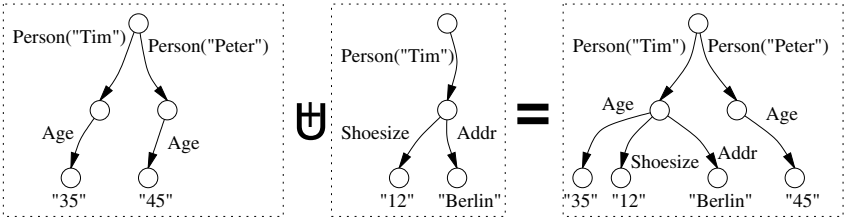


Fig. 3. Deep Union

Fig. 3 shows how the shoesize and address value for Tim can be added to a database with Tim and Peter.

Relational Databases in WHAX. There is a natural translation from relational databases into WHAX: Each tuple is represented by an outgoing edge from the root. The relation name R and the key k of the tuple form the local identifier $\mathsf{R}(k)$. All non-key attributes form a subtree under $\mathsf{R}(k)$.

3 Defining Views in WHAX

Over the past few years, several query languages for semi-structured and XML data have been proposed [12, 9, 3, 1]. Our language, called WHAX-QL, differs from XML-QL in that local identifiers (i.e. labels and keys) can be matched against patterns. We start by illustrating WHAX-QL through some examples.

View V_1: Select the name and the age of all authors older than 36:

```
V1($db) =  where      <Person($n).Age> $a </> in $db,
                      $a > 36
           construct  <MyPerson($n).Age> $a </>
```

The path expression Person(n).Age identifies the person's age in the database $\$db$ and binds the person's name to $\$n$. The subtree under Age is bound to variable $\$a$. Recall that the age is represented as a single outgoing edge from the Age node. To satisfy $\$a > 36$, the value of $\$a$ must be of the form $\{l : \{\}\}$ where l is an integer with $l > 36$.

View V_2: For each author, return the title, conference, and page numbers of publications:

```
V2($db) =
  where     <Conf(@name:$n,@year:$y).Publ(Title:$t)>
                <Author($a)> </>
                <Pages> $p </>
            </> in $db
  construct <Author(Name:$a).Publ(Title:$t,Conf:$n,Year:$y).Pages> $p </>
```

Variable $\$a$ binds to any author name and $\$p$ is bound to the pages of the publication identified by $\$n$, $\$y$, and $\$t$. The view is a tree with authors at the root and their publications at the leaves.

It is always possible to "unnest" nested variable bindings. For example, View V_2 could be equivalently written as:

[2] $\mathsf{dom}(v)$ is the set of local identifiers in v: $\mathsf{dom}(v) := \{l(k) \mid l(k) : s \in v\}$.

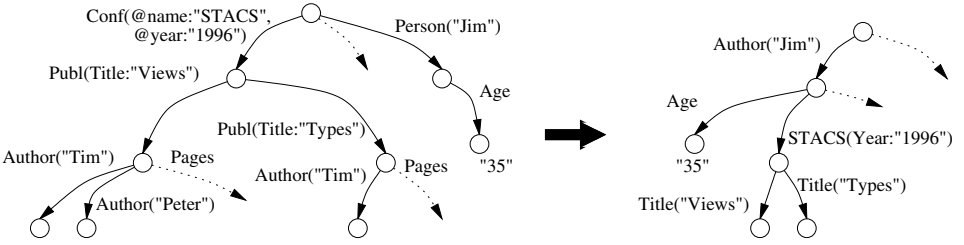


Fig. 4. Regrouping result for view V_3

$$
\begin{array}{lll}
\mathtt{e} & ::= & \text{"}str\text{"} \mid \$x \mid \mathtt{e}_1 \text{ op } \mathtt{e}_2 \mid \mathtt{e}_1 \uplus \mathtt{e}_2 \mid Q \mid (LPat(\mathtt{e}_1) : \mathtt{e}'_1, ..., LPat_n(\mathtt{e}_n) : \mathtt{e}'_n) \\
Q & ::= & \textbf{where} \quad <PPat_1> \$x_1 </> \textbf{ in } \$d_1, \ ... \\
 & & \qquad <PPat_m> \$x_m </> \textbf{ in } \$d_m, \\
 & & \qquad cond_1, ..., cond_n \\
 & & \textbf{construct} \ <PExpr_1> \mathtt{e}_1 </>, \ ..., \ <PExpr_p> \mathtt{e}_p </> \\
PPat & ::= & LPat_1(KPat_1). \ ... \ .LPat_n(KPat_n) \\
LPat & ::= & l \mid \$x \\
KPat & ::= & \$x \mid (l_1(\mathtt{v}_1) : KPat_1, ..., l_n(\mathtt{v}_n) : KPat_n) \\
PExpr & ::= & LPat_1(\mathtt{e}_1). \ ... \ .LPat_n(\mathtt{e}_n)
\end{array}
$$

Fig. 5. Syntax of WHAX-QL

```
V'2($db) =
  where     <Conf(@name:$n,@year:$y).Publ(Title:$t).Author($a)> </> in $db,
            <Conf(@name:$n,@year:$y).Publ(Title:$t).Pages> $p </> in $db
  construct <Author(Name:$a).Publ(Title:$t,Conf:$n,Year:$y).Pages> $p </>
```

As in XML-QL, multiple occurrences of the same variable represent the same value. In the example above, this ensures that author *$a* and pages *$p* belong to the same publication.

Example View V_3: For each person, return their age and all STACS publications. Furthermore, group all publication by their year:

```
V3($db) =
where     <Person(Name:$n).Age> $a </> in $db,
          <Conf(@name:"STACS",@year:$y).Publ(Title:$t).Author($n)></> in $db
construct <Author(Name:$n).Age> $a </>,
          <Author(Name:$n).STACS(Year:$y).Title($t)> </>
```

This query performs a join between persons and authors over variable $\$n$. Fig. 4 shows the regrouping effect of this query.

The Syntax of WHAX-*QL* (Fig. 5). The where-clause of Q describes the *variable bindings*. For each valuation of the variables in the where-clause, the construct-clause is evaluated and the results are deep-unioned together.

The identifiers $\$d_1, \ ..., \ \$d_n$ in the where-clause denote WHAX data sources and are called *base variables*. A *path pattern* $PPat_i$ matches a path in $\$d_i$ against a given sequence of *label patterns* $LPat$ and *key patterns* $KPat$, separated by dots: $LPat_1(KPat_1). \ ... \ .LPat_n(KPat_n)$. The value at the end of the path is bound to variable $\$x_i$. As in XML-QL, variable $\$x_i$ can be omitted if it is not used.

A label pattern *LPat* is either a constant label l or a label variable $\$x$, and a key pattern *KPat* is either a variable $\$x$ or a complex pattern $(l_1(\mathsf{v}_1) : KPat_1, ..., l_n(\mathsf{v}_n) : KPat_n)$. It matches a WHAX-tree if the tree has exactly n elements with distinct local identifiers $l_1(\mathsf{v}_1)$, ..., $l_n(\mathsf{v}_n)$ and the element values match patterns $KPat_1$, ..., $KPat_n$, respectively. If *KPat* (or v) in *LPat*(*KPat*) ($l(\mathsf{v})$, respectively) is the empty value $\{\}$, then we write *LPat* (l, respectively).

Variables in WHAX-*QL.* We distinguish several types of variables by their first occurrence within a WHAX-expression. Parameters to the query (such as *$db*) are called *parameter variables.* Variables in path patterns $PPat_i$ are bound to labels or key values and are called *label variables* and *key variables*, respectively. Variables $\$x_i$ at the leaves of path bindings are called *value variables.*

Syntactic Simplifications Views V_2 and V_2' in Sec. 3 illustrate how nested bindings can be eliminated. Similarly, we can eliminate the case where some value variable $\$x_i$ from $<PPat_i> \$x_i </>$ **in** $\$db_i$ is the base variable $\$db_j$ of some other pattern in the same where-clause: $<PPat_j> \$x_j </>$ **in** $\$x_i$. The second pattern can be replaced by pattern $<PPat_i.PPat_j> \$x_j </>$ **in** $\$db_i$.

4 View Maintenance through Multi-Linearity

The multi-linearity law is the foundation for efficient incremental view maintenance of bulk updates. For example, consider a multi-linear view $V(R_1, R_2)$: $V(R_1 \cup \triangle R_1, R_2) = V(R_1, R_2) \cup V(\triangle R_1, R_2)$ and $V(R_1, R_2 \cup \triangle R_2) = V(R_1, R_2) \cup V(R_1, \triangle R_2)$. The updated view can be easily computed by inserting the result of query $V(\triangle R_1, R_2)$ (or $V(R_1, \triangle R_2)$) to the view $V(R_1, R_2)$.

Unfortunately, WHAX-QL queries are not necessarily multi-linear with respect to deep union in their given form. We identify two properties to make WHAX-QL queries multi-linear: First, the *key variable constraint* forbids the use of parameter and value variables as key and operand variables. Second, the *base variable constraint* forbids the multiple use of base variables.

Key Variable Constraint. Recall example view V_1 from Sec. 3, which accesses value variable *$a* in condition $\$a > 36$. Consider the (rather unusual) insertion of a second age for some author. Then the condition will become false because *$a* is no longer a singleton tree. Hence, the view is not multi-linear.

Similarly, consider the mixed use of parameter/value variable *$x* as a key variable in a path pattern *PPat* (or path expression *PExpr*). The value of the parameter/value variable can change during an update and this will make *PPat* (*PExpr*) refer to a *completely different* value. Again, those views are not multi-linear. Therefore, we syntactically restrict WHAX-QL queries as follows:

Definition 1. *A* WHAX-*QL query is* maintainable *if no parameter/value variable $\$x$ occurs as a key variable or operand of some base operation* e_1 op e_2.

One can sometimes replace the query by a similar query that returns the same expected result. For example, view V_1 can be transformed by binding $\$a$ to the label (i.e. the age value) of each Age edge: This multi-linear view has the same intended result.

```
V''1($db) =  where      <Person($n).Age.$a> </> in $db,
                        $a > 36
             construct  <MyPerson($n).Age.$a> </>
```

Base Variable Constraint. View $V_3(\$db)$ in Sec. 3 uses variable $\$db$ twice as a base variable. One can observe that the view is *not* multi-linear for a database DB and update $\triangle DB$ (i.e. $V_3(DB \uplus \triangle DB) \neq V_3(DB) \uplus V_3(\triangle DB)$), since some persons or authors in $\triangle DB$ might join with authors (persons, resp.) in DB.

Fortunately, the view can be made multi-linear by replacing the second occurrence of $\$db$ with a fresh variable $\$db'$ and adding $\$db$ as a parameter variable:

```
V'3($db,$db') =
 where     <Person(Name:$n).Age> $a </> in $db,
           <Conf(@name:"STACS",@year:$y).Publ(Title:$t).Author($n)></> in $db'
 construct <Author(Name:$n).Age> $a </>,
           <Author(Name:$n).STACS(Year:$y).Title($t)> </>
```

The new multi-linear view $V_3'(\$db, \$db')$ is equivalent to $V_3(\$db)$, if applied to the same database: $V_3'(DB, DB) = V_3(DB)$.

Theorem 1. *A maintainable* WHAX-*QL view* $V(\$d_1, \ldots \$d_n)$ *is multi-linear in its parameters* $\$d_1, \ldots \$d_n$ *if all base variables* $\$db$ *of the same* where-construct-*expression are distinct and do not occur in the* construct-*clause.*

Lemma 1. *A maintainable view* $V(\$db_1, \ldots, \$db_n)$ *can always be transformed into an equivalent multi-linear view.*

Details of the rewriting process can be found in [17].

Relational SPJ Views and WHAX-*QL Views.* Based on the mapping in Sec. 2, it can also be proven that WHAX-QL is a generalization of relational SPJ-queries:

Lemma 2. *Relational SPJ views can be expressed as maintainable* WHAX *queries.*

5 Deletions

Multi-linearity cannot be used for propagating deletion updates. The reason is analogous SPJ-views: a tuple in the view may have multiple derivations from tuples in the base relations. Hence, it is unclear when the tuple should be deleted.

Two approaches to deletions have been studied for SPJ-views: 1) *View analysis* [6] and 2) view maintenance using *multi-set semantics* [14] or *counting* [4, 15]. The view analysis algorithm in [6] is a static decision procedure which accepts only views for which any tuple in the view is guaranteed to have exactly one derivation from the base relations. Alternatively, one can keep track of the number of derivations for each distinct tuple in the view [14, 4, 15].

These techniques are based on invertible versions of the set union operation: *disjoint*, *bag*, or *counting* union. Intuitively, invertible union operation allow the efficient propagation of deletions since the following multi-linearity law for deletion can be derived: $V(D_1, \ldots, D_i - \bigtriangledown D_i, \ldots, D_n) = V(D_1, \ldots, D_i, \ldots, D_n) - V(D_1, \ldots, \bigtriangledown D_i, \ldots, D_n)$.

Unfortunately, the deep union operation in WHAX is not invertible. Furthermore, the natural disjoint union operation for WHAX is complicated to handle (preliminary ideas can be found in [17]). Instead, we focus on the counting technique in the next section.

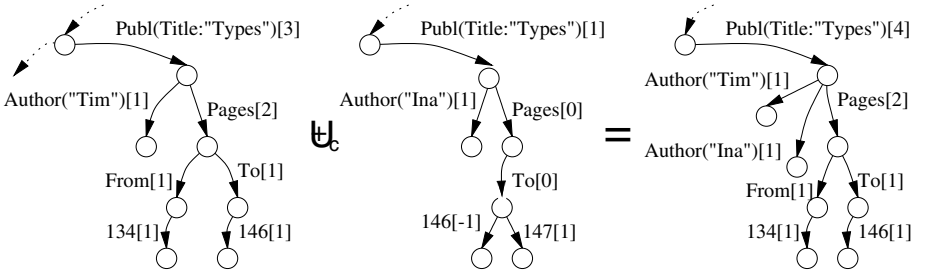


Fig. 6. Annotating a WHAX-tree with supports

5.1 Counting in WHAX

Similar to counting for SPJ-views [4, 15], each edge with local identifier $l(k)$ in the WHAX-tree is annotated with a count c that describes the number of derivations for the edge: $l(k)[c]$. We call count c the *support of the edge*.

The tree on the left of Fig. 6 shows parts of the tree from Fig. 2 with supports. Leaf edges have support 1, and the support of the inner edges is the sum of the children's supports (the *indirect support* of the edge) and some (by default, zero) *direct support* for the edge itself. The indirect support of leaf edges is 0.

Insertions and deletions are represented as trees with positive or negative supports. Fig. 6 shows how a new author "Ina" is added to publication "Types" and how a page number is changed from 146 to 147. Note that the change is modeled as a deletion (support -1) plus an insertion (support 1). The deep union operator $\uplus_c$ for trees with counts is defined as follows:

$$
\begin{array}{rl}
v_1 \uplus_c v_2 ::= \{l(k)[c_1 + c_2] : (s_1 \uplus_c s_2) & | \; l(k)[c_1] : s_1 \in v_1, l(k)[c_2] : s_2 \in v_2, \\
 & \quad (c_1 + c_2 \neq 0 \;\vee\; s_1 \uplus_c s_2 \neq \{\}) \;\} \cup \\
\{l(k)[c] : s & | \; l(k)[c] : s \in v_1, l(k)[...] \notin \mathsf{dom}(v_2)\} \cup \\
\{l(k)[c] : s & | \; l(k)[c] : s \in v_2, l(k)[...] \notin \mathsf{dom}(v_1)\}
\end{array}
$$

Note that merged edges are eliminated if their support is empty ($c_1 + c_2 = 0$) and if they do not have children ($s_1 \uplus_c s_2 = \{\}$).

The supports in the view must be carefully chosen to preserve the semantics of queries, as the following nested view illustrates:

```
where     <Person(Name:$n)> $p </> in $db
construct <MyPers(Name:$n)>
             where <Age> $a </> in $p  construct <Age> $a </> s </>
```

The *indirect* support of path MyPers(Name:n) in the view may be 0 if there is no Age edge in the base data. Its *direct* support must therefore be > 0, and is determined by the (direct+indirect) support of Person(Name:n) in the source.

Consider the syntax of where-construct-queries (Fig. 5). Let ϕ denote a valuation for the variables in the where-clause and let $PPat_i(\phi)$ ($PExpr_j(\phi)$) be the instantiation of pattern $PPat_i$ (path expression $PExpr_j(\phi)$, respectively) under this valuation. Let the support of path p, $Supp(p)$, be the the support (direct + indirect) of the *last* edge in the path. The direct support of a path ($DSupp(p)$) is defined similarly. Then, the direct support of each output path $PExpr_j(\phi)$ is determined by the product of the supports of each of the input paths $PPat_i$: $\forall\, 1 \leq j \leq n \;:\; DSupp(PExpr_j(\phi)) = \prod_{1 \leq i \leq m} Supp(PPat_i(\phi))$.

Lemma 3. *Each multi-linear* WHAX*-QL view* $V(\$d_1, \ldots \$d_n)$ *is also multi-linear under counting semantics using counting deep union* $\uplus_c$.

6 Aggregations in WHAX

A simple extension of WHAX allows the use of aggregations. First, the syntax is extended such that expressions e_i in the construct-clause of where-construct-queries (Fig. 5) can be any aggregate function sum(e), avg(e), count(), min(e), or max(e). The following view determines the number of pages published for each conference and year:

```
where     <Conf(@name:$cn,@year:$cy).Publ(Title:$t).Pages>
                <From> $from </> <To> $to </> </> in $db,
construct <Conf(@name:$cn,@year:$cy).SumPages> sum($to-$from+1) </>
```

Aggregate values produced by the construct-clause are merged through *aggregate deep union* $\uplus_a$ with the following changes in the model: Leaf edges with annotation $l[c]$[3] can be annotated with aggregate tags: aggr :: $l[c]$. For example, Count::2[2] represents a publication count of 2 (with support 2) and min::24[1] a minimum author age of 24.

The deep union $\uplus_a$ over WHAX-trees v_1 and v_2 with aggregates has the following meaning: For any two matching aggregate elements aggr :: $l_1[c_1] \in v_1$ and aggr :: $l_2[c_2] \in v_2$, a new edge with aggr :: $l[c_1 + c_2]$ with $l = \text{aggr}(l_1, l_2, c_1, c_2)$ is created. The aggregate combinator function aggr(l_1, l_2, c_1, c_2) is defined depending on aggr: sum$(l_1, l_2, c_1, c_2) = l_1 + l_2$, min$(l_1, l_2, c_1, c_2) =$ min(l_1, l_2), etc. Note that min and max are undefined for $c_1 * c_2 < 0$, since min and max cannot be incrementally maintained under deletions.

7 Conclusion and Future Work

The WHAX-model is a hierarchical data model similar to the deterministic model [5] and LDAP [18]. In contrast to previous work on view maintenance in semistructured databases [2, 19], the WHAX architecture supports complex restructuring operations such as regrouping, flattening, and aggregations. Furthermore, while existing work only considers atomic updates, our technique is applicable to large batch updates, which can reduce the refresh time of warehouses considerably. The underlying multi-linearity principle is also important for other optimization problems, such a query parallelization and pipelining.

Our approach can be extended with other view maintenance techniques, e.g. deferred view maintenance [11] or detection of irrelevant updates [4].

The WHAX model does not currently support ordered data structures. Ordered structures are difficult to maintain if updated elements are identified over their position. Positions are "dynamic" in two ways: 1) positions of elements might change during updates, and 2) positions may be different in views. Therefore, a mapping between "dynamic" positions and "static" keys must be provided. This mapping changes during updates and efficient index schemes (e.g. based on B-trees) are required.

Acknowledgements: We would like to thank Peter Buneman and Wang-Chiew Tan for helpful discussions.

[3] The key k and the value v in $l(k) : v$ are always empty for aggregate edges.

References

1. S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web - From Relations to Semistructured Data and XML.* Morgan Kaufmann Publishers, San Francisco, California, 1999.
2. S. Abiteboul, J. McHugh, M. Rys, V. Vassalos, and J. Wiener. Incremental maintenance for materialized views over semistructured data. In *Int'l Conference on Very Large Databases (VLDB)*, pages 38–49, New York City, NY, August 1998.
3. S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. Wiener. The Lorel query language for semistructured data. *International Journal on Digital Libraries*, 1996.
4. J.A. Blakeley, N. Coburn, and P.-A. Larson. Updating derived relations: Detecting irrelevant and autonomously computable updates. *ACM Transactions on Database Systems*, 14(3):369–400, September 1989.
5. P. Buneman, A. Deutsch, and W.C. Tan. A deterministic model for semi-structured data. In *Workshop on Query Processing for Semistructured Data and Non-Standard Data Formats*, Jerusalem, Israel, January 1999.
6. S. Ceri and J. Widom. Deriving production rules for incremental view maintenance. In *17th Int'l Conference on Very Large Data Bases (VLDB)*, pages 577–589, Barcelona, Spain, September 1991. Morgan Kaufmann.
7. S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom. The TSIMMIS project: Integration of heterogenous information sources. In *Proceedings of the Information Processing Society of Japan Conference*, Tokyo, Japan, October 1994.
8. V. Christophides, S. Cluet, and J. Simeon. On wrapping query languages and efficient XML integration. In *ACM SIGMOD Conference on Management of Data*, 2000. (to appear).
9. J. Clark. XSL transformations (XSLT), version 1.0. *W3C Proposed Recommendation*, Octover 1999. Available as `http://www.w3.org/TR/xslt`.
10. J. Clark and S. DeRose. XML path language (XPath), version 1.0. *W3C Working Draft*, August 1999. Available as `http://www.w3.org/TR/xpath`.
11. L.S. Colby, T. Griffin, L. Libkin, I.S. Mumick, and H. Trickey. Algorithms for deferred view maintenance. In *1996 ACM SIGMOD Conference*, pages 469–480, Montreal, Canada, June 1996.
12. A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu. XML-QL: a query language for XML. *W3C Note Note-xml-ql-19980819*, 1998. Available as `http://www.w3.org/TR/NOTE-xml-ql`.
13. D. Gluche, T. Grust, C. Mainberger, and M. H. Scholl. Incremental updates for materialized OQL views. *Lecture Notes in Computer Science (LNCS)*, pages 52–66, December 1997.
14. T. Griffin and L. Libkin. Incremental maintenance of views with duplicates. In *ACM SIGMOD Conference*, pages 328–339, San Jose, California, May 1995.
15. A. Gupta and I.S. Mumick. Maintenance of materialized views: Problems, techniques, and applications. *IEEE Data Engineering Bulletin*, 18(2):3–18, June 1995.
16. H.A. Kuno and E.A. Rundensteiner. Incremental maintenance of materialized object-oriented views in multiview: Strategies and performance evaluation. *IEEE Transactions on Knowledge and Data Engineering*, 10(5), 1998.
17. H. Liefke and S.B. Davidson. Efficient view maintenance in XML data warehouses. Technical Report MS-CIS-99-27, Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA 19104, November 1999.
18. M. Wahl, T. Howes, and S. Killes. Lightweight directory access protocol (v3). Technical report, IETF, December 1997.
19. Y. Zhuge and H. Garcia-Molina. Graph structured views and their incremental maintenance. In *14th Int'l Conference on Data Engineering (ICDE)*, pages 116–125, Orlando, Florida, February 1998.

Maintaining Horizontally Partitioned Warehouse Views

Mei Xu
School of Computer Science, University of Windsor
C.I. Ezeife*
School of Computer Science University of Windsor,
Windsor, Ontario, Canada N9B 3P4
cezeife@uwindsor.ca

Abstract

Data warehouses usually store large amounts of information, representing an integration of base data from different data sources over a long time period. Aggregate views can be stored as a set of its horizontal fragments for the purposes of reducing warehouse query response time and maintenance cost.

This paper proposes a scheme that efficiently maintains horizontally partitioned data warehouse views. Using the proposed scheme, only one view fragment holding the relevant subset of tuples of the view is accessed for each update. The scheme also includes an approach to reduce the refresh time for maintaining views that compute aggregate functions MIN and MAX.

Keywords: Data Warehouse Applications, View Maintenance, Horizontal Partitioning, Performance Improvement.

1 Introduction

Different data sources are usually integrated into one large central fact table in the data warehouse holding the main integrated data. Foreign key attributes in the fact table may refer to dimension tables, which describe the dimension attributes in the fact table. Data warehouse views are derived from the fact table and may possibly join one or more of the dimension tables. Cells of an n-dimensional data cube hold data that can easily construct 2^n subviews for an aggregate measure of interest. Breaking up a relation into its horizontal fragments would lead to a reduction in both query response time and maintenance

*This research was supported by the Natural Science and Engineering Research Council (NSERC) of Canada under an operating grant (OGP-0194134) and a University of Windsor grant.

Y. Kambayashi, M. Mohania, and A M. Tjoa (Eds.): DaWaK 2000, LNCS 1874, pp. 126–133, 2000.
© Springer-Verlag Berlin Heidelberg 2000

cost. Horizontal fragmentation partitions a relation such that each fragment contains only a subset of tuples of the relation that satisfy specific predicate conditions [2].

Data in a view may be deleted, updated, or new data can be inserted into a view with each update expressed as a delete-insert pair. These constitute refresh operations. Views can be maintained by computing the changes made to data sources and applying these changes to views to bring them up-to-date without accessing base data. This maintenance method is called incremental view maintenance.

1.1 Related Work

Harinayaran et al. in [3] uses a lattice framework to model the dependencies between subviews of the data cube, and also presents a greedy algorithm for selecting some of the views to store. Ezeife in [2] presents a scheme that stores a selected view as a set of its horizontal fragments to reduce query response time and maintenance cost.

Work on incremental view maintenance techniques include [1, 4]. Colby *et al.* in [1] proposes to split the deferred maintenance work into propagate and refresh functions in order to minimize the batch time needed for maintenance. Mumick *et al.* in [4] shows how the above two functions can be derived for the aggregate views.

1.2 Contributions

This paper contributes by proposing a new view maintenance scheme for horizontally partitioned data warehouse cube views which leads to reduced view maintenance cost. The scheme handles all three types of refresh operations. In addition, this paper proposes a method to reduce the refresh time for views with aggregate functions MIN and MAX by eliminating the need to visit the base relations in order to compute a new minimum or maximum value when the current one is deleted from base table. The proposed new scheme and the schemes presented in [4] are implemented and compared.

1.3 Outline of the Paper

Section 2 gives an example to show how to maintain a horizontally partitioned view using the proposed approach. Section 3 formally presents the new scheme for maintaining horizontally partitioned data cube views. Section 4 discusses two experimental study cases to demonstrate the performance and benefits of the new scheme, while section 5 presents conclusions and future work.

2 An Example

The following is an example of a data warehouse for the service departments of a car dealer chain company. This company has many branches at different

Table 1: Car Dealer Fact Table DFT

DealerID	ProviderID	ServiceTime	Gain
003	02	199904020030	200
001	02	199904020048	300
002	01	199904020049	500
002	03	199904020050	400
001	01	199904020068	200
001	01	199904020064	50
003	02	199904020067	-200
002	03	199904020070	600
002	01	199904030070	200
001	02	199904030080	200
001	02	199904030090	100

locations. The data warehouse has the following fact and dimension tables:
DFT (DealerID, ProviderID, ServiceTime, Gain)
Dealer (DealerID, DealerName)
Provider (ProviderID, ProviderName)
Time (ServiceTime, Month, Year)

The attributes of the fact table DFT consists of a dealer's branch ID, service provider's ID, service time in minute, and the gain or profit of a single service. Each tuple in the fact table contains the gain of a single service provided by a service provider in a dealer branch at a certain time. If the gain is negative, it means that a customer is not satisfied with a provided service and has asked for a refund. The ServiceTime contains twelve digits, e.g., 199904020030. The first four digits indicate the service year, and the second four digits indicate the service month and day. The last four digits in ServiceTime represent the service minute of the day starting from 0 and ending with 1439. The table DFT (Table 1) contains sales information for the chain company from minute 0030 April 2 to minute 0090 April 3, 1999.

Assume that only the top-level aggregate view (DPM) is stored in the data warehouse. D stands for DealerID, P stands for ProviderID, and M stands for service Month. DPM records the monthly total service amount, minimum gain, and the total numbers of non-refunded services for each individual service provider in each dealer branch. In order to handle deletions in some situations, an SQL aggregate function Count(*) is added to the view. DPM has the following schema: DPM (DealerID, ProviderID, ServiceMonth, Total, MinGain, TotalCount). The top view DPM is shown in Table 2.

With the example, assume that $DealerID \leq 001$, *and* $ProviderID > 02$ are the simple predicates SP_1, SP_2 from queries accessing the warehouse view DPM, that the fragmentation process was based on. Minterm predicates are used to partition the view and the four minterm predicates created using these two simple predicates by obtaining their conjunctions in either their natural or

Table 2: DPM View (V1)

DealerID	ProviderID	Service	Total	MinGain	Count(*)
001	01	199904	250	50	2
001	02	199904	600	100	3
002	01	199904	700	200	2
002	03	199904	1000	400	2

Table 3: Net-Change-DPM Table

DealerID	ProviderID	Service	Total	MinGain	Count(*)
001	01	199904	100	200	1
001	03	199904	400	400	1
001	02	199904	-100	100	-1
002	01	199904	-700	200	-2

negated form [2] are:
$M_1 = SP_1 \wedge SP_2 \Rightarrow DealerID \leq 001 \wedge ProviderID > 02$
$M_2 = SP_1 \wedge \neg SP_2 \Rightarrow DealerID \leq 001 \wedge ProviderID \leq 02$
$M_3 = \neg SP_1 \wedge SP_2 \Rightarrow DealerID > 001 \wedge ProviderID > 02$
$M_4 = \neg SP_1 \wedge \neg SP_2 \Rightarrow DealerID > 001 \wedge ProviderID \leq 02$
These four created minterm predicates define the four fragments of DPM. The changes (services and refunds) made to the fact table DFT are recorded in two tables Insertion-DFT (for all inserts) and Deletion-DFT (for all deletes). Services rendered by (DealerID, ProviderID) are (001, 01) with a gain of $300 and (001, 03) with a gain of $400. Refunds done are as follows: (001, 02) received a service gain of $-100 through refund, (001, 01) got $-200, (002, 01) got $-200 while (002, 01) had $-500.

To compute the changes to be made to DPM (top view), three virtual views Insertion-DPM, Deletion-DPM and Insert-Delete-DPM are created. Insertion-DPM, Deletion-DPM are derived from Insertion-DFT and Deletion-DFT respectively using the same scheme as top view DPM derived from fact table DFT. The Insertion-DPM and Deletion-DPM are unionized to create the Insert-Delete-DPM virtual view. The net-change table, Net-Change-DPM (Table 3) is created by aggregating Insert-Delete-DPM. Each tuple in a net-change table causes a single refresh in the corresponding view DPM. The view being used for refresh operation is called the Current View, while the tuple of a net-change table, currently being used to refresh the Current View is called the Active Tuple. For each Active Tuple in a net-change table, the algorithm finds the Needed Fragment and the Matched Tuple and updates the Matched Tuple according to the Active Tuple. If the Matched Tuple is not found, it means the Active Tuple in the net-change table is new to the Needed Fragment, then it is inserted into the Needed Fragment.

In order to reduce the refresh time for recomputing the new value from

Table 4: Min-DPM

RowId	001-02-9904	002-01-9904	002-03-9904	001-01-9904
1	200	500	600	200

Table 5: The Updated Fragments of DPM

Frag	DealerID	ProviderID	Service	Total	MinGain	Count(*)
F_1(M1)	001	03	199904	400	400	1
F_2(M2)	001	01	199904	350	50	2
	001	02	199904	500	200	2
F_3(M3)	002	03	199904	1000	400	2
F_4(M4)	none					

fact table for the aggregate function MIN in DPM when the current minimum value is deleted, we create a view Min-DPM (Table 4) derived from the fact table DFT for view DPM. It contains the next smallest gain from the updated fact table for each tuple (different sets of group-by attribute values) in DPM. The updated fragments of DPM are presented in Table 5.

3 The Proposed Partitioned View Maintenance Schemes (PVMS)

The partitioned view maintenance scheme (PVMS) consists of a main algorithm that takes horizontally partitioned warehouse data cube views and the changes made to the fact table as input, and returns the updated horizontally partitioned cube views. The approach assumes that the fact table has been updated. In order to make warehouse data more available to users, the strategy of splitting the maintenance work into propagate and refresh functions [1, 4] is adopted in this algorithm.

The two functions are called from the main algorithm PartitionedView-Maintenance. The PartitionedView-Maintenance algorithm first implements the Propagate function routine through the functions aggregate-changes-for-topview and aggregate-changes-for-View. The formal presentation of the main algorithm PartitionedViewMaintenance is given as Figure 1 and further details on both Propagate and Refresh functions are discussed next.

3.1 Propagate Function

In algorithm PartitionedView-Maintenance, the called function aggregate-changes-for-Topview, creates a net-change table for only the top view, V_1 from changes made on the fact table. The net-change table is used later to update the top view. In order to create the net-change table for the top view, all changes on fact table are expressed as insertion and deletion pairs. Insertions are recorded in a table called Insertion-FT, and deletions are recorded in another table called

Algorithm 3.1 *(PartitionedView-Maintenance - Maintaining horizontally partitioned views)*

Algorithm PartitionedView-Maintenance()
Input: set of fragments F_m of horizontally partitioned data cube views V_i, for each view, changes C_j from fact table
Output: a set of updated horizontally partitioned data cube views
begin
// 1. Propagate function: count the net changes for each partitioned view //
Net-Change-V_1 = aggregate-changes-for-topview(V_1, C_j) //computes for top view //
// Creating the net-change table Net-Change-V_i for other views in the lattice //
for i = 2 to number of views in the lattice
Net-Change-V_i = Aggregate-changes-for-view(view V_i, Net-Change-V_s)
// 2. Refresh function: refresh the data cube partitioned view using the //
for i = 1 to number of views in the lattice
refreshed partitioned view V_i = Refresh-View(V_i, Net-Change-V_i)
end // of PVM //

Figure 1: The PartitionedView-Maintenance Algorithm

Deletion-FT. The two tables have the same heading format as the fact table. Insertion-V_1 and Deletion-V_1 projected from Insertion-FT and Deletion-FT respectively using the same scheme as the top view derived from the fact table, are created next. Unionizing Insertion-V_1 and Deletion-V_1 creates a virtual view called Insert-Delete-V_1. The net-change table Net-Change-V_1 for top view is created from Insert-Delete-V_1 after aggregating.

Function aggregate-changes-for-View is used to create a net-change table called Net-Change-V_i for each data cube partitioned view, V_i, that is not top view.

3.2 Refresh Function

Function Refresh-View is designed for refreshing each horizontally partitioned view in the data cube by applying changes in the corresponding net-change table computed from the propagate function. It takes all the fragments of a partitioned view and the corresponding net-change table as input and outputs the refreshed view partitions. The summary of steps involved in the function Refresh-View called by algorithm PartitionedView-Maintenance is given below. For each Active Tuple in the net-change table, do the following: (1) Find the Needed Fragment for the Active Tuple, (2) Find the Matched Tuple in the View Fragment. Lastly, update or delete the Matched Tuple according to the information in the Active Tuple. If there is no Matched Tuple, insert the Active Tuple in the Needed Fragment. The sum of count columns of both the Active Tuple (AT_t) and the Matched Tuple (T_j), expressed as $AT_t.Count(*)$ and $T_j.Count(*)$ is used to decide if the refresh is going to delete the Matched Tuple T_j or update

the aggregate attribute values in the T_j. If the sum is zero, the function deletes the T_j, otherwise, it updates the aggregate attribute values.

3.3 Other Added Features of The PVM Algorithm

Aggregate functions SUM and COUNT are self-maintainable with respect to insertion and deletion. Therefore, the new value of SUM or COUNT can be computed by adding $AT_t.Sum(exp)$ or $AT_t.Count(exp)$ and $T_j.Sum(exp)$ or $T_j.Count(exp)$ together. However, MIN and MAX are self-maintainable with respect to insertion, but not to deletion. Thus, before updating the aggregate attribute values of T_j, the Refresh function checks if it is necessary to re-compute the new minimum or maximum value. If Min(exp) of AT_t is less than or equal to Min(exp) of T_j, or Max(exp) of AT_t is greater than or equal to Max(exp) of T_j, the new value of Min(exp) or Max(exp) in T_j will need to be re-computed from the base table which is time consuming. To reduce time for re-computing the new minimum or maximum from base tables, the PVM algorithm uses a smaller view called MinMax-View for each partitioned view containing aggregate function MIN or MAX. A MinMax-View contains the first k smallest and/or the first k largest values from the changed base relations for each group-by set of tuples in the corresponding view. The value of k is determined by data warehouse administrator according to the application.

4 Performance Analysis

Implemented are the proposed PartitionedView-Maintenance Algorithm (PVM-algorithm) and the scheme in [4] called MQM-algorithm. The CPU execution time (measured in seconds), for maintaining a partitioned view is taken as the only maintenance cost in the analysis. After some preliminary runs of both schemes on an Oracle 8 DBMS on a Pentium 400MHz IBM PC platform, it was seen that the performance of the proposed scheme was affected by both the size and the partitioning criteria of a view. Therefore, the following two experimental study cases were designed.

Case 1: Given a fixed size horizontally partitioned data warehouse view, different partitioning criteria were used to obtain different numbers of fragments for the view and both the PVM and MQM algorithms were compared.

Case 2: Given a horizontally partitioned view with varying sizes and same partitioning criterion, both PVM-algorithm and MQM-algorithm were run.

Table 6 shows the CPU execution time using PVM-algorithm for maintaining view V under four different partitioning criteria involving only insertion, deletion, or update operations. Since insertion takes high percentage of maintenance time, the contribution of the PVM algorithm, which drastically reduces this time is tangible. Table 7 tabulates the CPU execution time of PVM-algorithm and MQM-algorithm for only four fragments in the view.

Table 6: Maintenance Time for the 10,000-row View

Type of Refresh	CPU Execution Time			
	1 fragment	2 fragments	4 fragments	8 fragments
Insertion	6966.5	3714.5	1837.0	1038.0
Deletion	4512.8	2264.5	1233.7	672.4
Update	4352.9	2570.0	1270.6	629.3

Table 7: Comparing CPU Maintenance Time for Varying View Sizes

Size of View	2000	4000	6000	8000	10000
PVM	374.2	615.8	931.7	1222.4	1538.9
MQM	1015.03	1870.4	2912.4	4052.7	5280.8

5 Conclusions and Future Work

This paper presents a scheme for maintaining horizontally partitioned data warehouse cube views efficiently since only one relevant fragment of the view is accessed for each maintenance operation and not the entire view. The proposed scheme handles all three types of refresh operations of deletion, insertion and update, and includes a technique for reducing the refresh time for maintaining views with aggregate function MIN and MAX. The experimental improvement ratios of the maintenance times using the proposed maintenance scheme in comparison to that of [4] is approximately 68% when the size of a view is between 20,000 and 100,000 rows and this ratio shows to increase steadily with increase in the size of the view as shown in Table 6.

Future work includes conducting experimental studies to measure (1) the benefits of storing MinMax-Views, and (2) the utility of this approach in comparison with indexing technique.

References

[1] Colby, L.S., Griffin, T., Libkin, L.: Algorithms for Deferred View Maintenance. Proceedings of the ACM SIGMOD International Conference, Montreal, Canada,(1996) 469-480

[2] Ezeife, C.I.: Selecting and Materializing Horizontally Partitioned Warehouse Views. to appear in Elsevier journal of Data and Knowledge Engineering, (2000).

[3] Harinarayan, Venky and Rajaraman, Anand, Ullman, Jeffret, : Implementing Data Cubes Efficiently. ACM SIGMOD International Conference on Management of Data, (1996), 205–216.

[4] Mumick, I.S., Quass, D., Mumick, B.S.: Maintenance of Data Cubes and Summary Tables in a Warehouse. Proceedings of the ACM SIGMOD International Conference, (1997) 100-111.

Funding Research in Data Warehousing and Knowledge Discovery EPROS : The European Plan for Research in Official Statistics

Jean-Louis Mercy and Harald Sonnberger
Eurostat, European Commission, L-2920 Luxembourg
e-mail : harald.sonnberger@cec.eu.int

Abstract: This paper discusses: (1) the challenges that the European Statistical System (ESS) faces as the result of the recent appearance of phenomena such as the information society and the new economy, and (2) the extent to which new technological developments in data warehousing, knowledge discovery and extensive use of the internet can contribute to successfully meeting these challenges. Two specific issues are considered: the network nature of the ESS, and the new ways of producing statistics that reinforce the needs for research applied to statistics in domains such as data integration, distributed databases, EDI, automated data capture, analytical tools and dissemination. A historical overview is given of research activities financed by the European Commission as well as their relevance for DaWaK2000. A primary goal of this paper is to provide information about relevant research within the European Statistical System, and to invite the scientific community to participate actively in upcoming calls for proposals and calls for tender financed under the IST programme to solve the urgent needs for timely and high-quality statistics.

1 Introduction

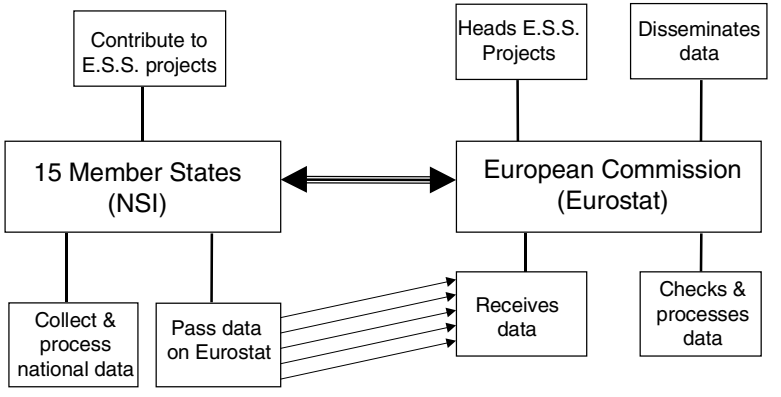
The challenges for the European Statistical System (ESS) have never been greater than they are at present. New phenomena, such as globalisation, the appearance of the information society, the effects of the "new economy" on employment and competitiveness, the creation of and the need for short-term indicators for the euro-zone, the on-going and up-coming negotiations for EU membership with the candidate countries, etc. have all increased the pressure on the ESS and in particular on Eurostat, to produce relevant, timely, and accurate statistical information.

The ESS responds to these challenges with two types of actions: first, by commissioning a large number of research projects (discussed in detail below); and second, by taking concrete first steps towards a common IT policy, which was in the past mainly been considered as a private matter of the National Statistical Institutes (NSIs). Under the pressure of increasing user demands for more and faster statistics (e.g. Internet, e-commerce, mobile phones) and given the difficulties of the traditional approach of NSIs promptly to respond to these needs, however, this attitude has changed. Given the similarity of concerns of all data providers, the necessity to organise the ESS as a network, the common pressure of an extremely dynamic technological environment, the European integration process, the scarcity of resources and the complexity of the problems, a common IT policy for the ESS seems indispensable to all ESS partners.

Y. Kambayashi, M. Mohania, and A M. Tjoa (Eds.): DaWaK 2000, LNCS 1874, pp. 134-145, 2000.
© Springer-Verlag Berlin Heidelberg 2000

Fig. 1: The European Statistical System (ESS):

A partnership between the Commission and the 15 members states



Various user satisfaction surveys have indicated that users demand higher quality. Data must be available on specific dates, to everyone. The Web is becoming the most natural way to access information, and information needs are broadening. Access to tabular data is no longer sufficient ; the research community wants disaggregated information. Common solutions could be considered, as illustrated by the interest of some Member States in the creation of Web sites and the implementation of data warehouses. NSIs are attempting to integrate their information systems (which, in the past, were primarily based on surveys), to make extensive use of administrative sources and to promote the use of standards in data exchanges.

Much has already been achieved: common R&D projects co-financed by Community funds; a high level of involvement of the ESS in standardisation activities (GESMES, CLASET); development of a common architecture to enable the distribution of statistical services (DSIS); common tools to exchange information (Stadium, Statel); common developments in sectors where the European integration is well advanced (Edicom); and the launching of a programme for the transfer of technology and know-how.

However, much remains to be done. In order to be prepared for future challenges, more in-depth co-operation in the field of IT is indispensable.. NSIs are independent organisations with different traditions and cultures. Their sizes differ and they are part of administrative systems with a legacy. Eurostat has recently created an IT Steering Committee that will address these issues. It will co-ordinate activities in the traditional areas of R&D, metadata and reference environments, data exchange and

transfer of technology and know-how. Efforts will continue to improve the harmonisation of metadata. The areas of e-commerce, data warehousing and metadata have been identified as topics of high interest and will be treated with priority.

2 The Statistical Production Process

Official statistics are difficult to define in terms of statistical methods employed. For the purpose of this presentation, it is more useful to look at the statistical production process which can be roughly divided into four parts: (1) methods and concepts definition; (2) data collection; (3) data processing and analysis; and (4) dissemination to end-users. The advantage of this process view is that the needs of official statistics can be defined in terms of process quality, and statistical and methodological techniques can be seen as tools for achieving the production goals. The needs within each of these phases is discussed below.

Methods and concepts definition: Although less significant for DaWaK, this phase is probably the most essential: without adequate and harmonised methods and definition of nomenclatures, units, variables, etc. no comparable statistics can be produced. Since official statistics were first collected, considerable efforts have been made to harmonise definitions at the world level. Nonetheless, a new conceptual approach is badly needed to describe the new information economy, its shape, structure and dynamics, the transition from the "old" economy and its implication in terms of technologies, impact on businesses, on the economy and on citizens.

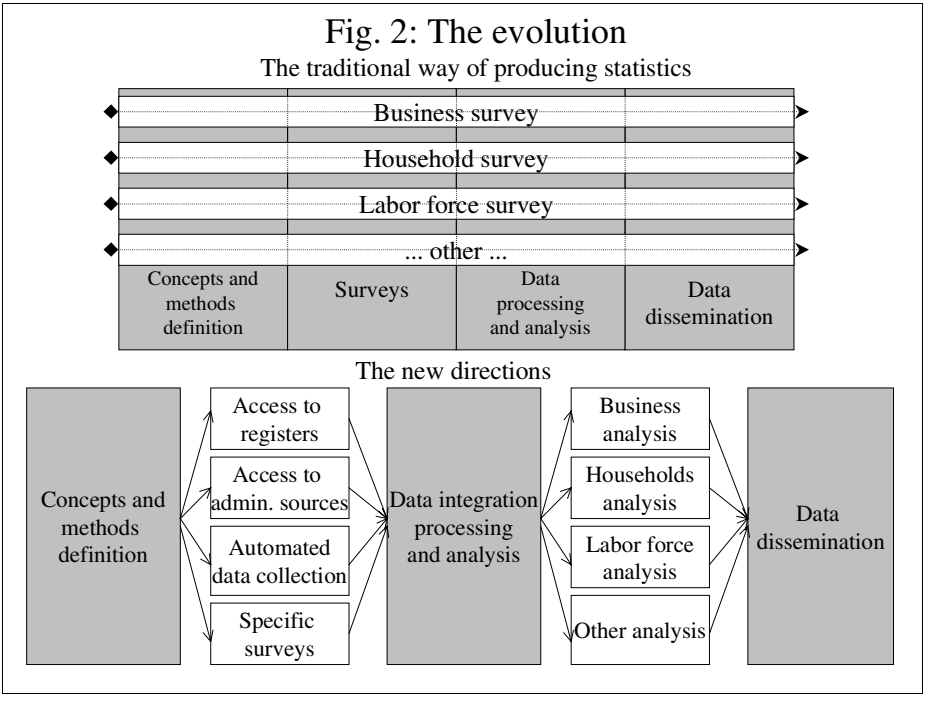
Data collection and data sources: The ESS is composed of Eurostat and the NSIs, with Eurostat being the focal point of pan-European statistical activities. Eurostat does not engage in data collection, as this is the job of the NSIs and their national partners. Eurostat's role is to check and guarantee the quality of data, including its accuracy, comparability, timeliness, coverage, etc. In Working Party meetings, Eurostat negotiates guidelines with the representatives of the Member States, designed to achieve the indispensable harmonisation of data collection (e.g. harmonised questionnaires in all languages for all Member States) and data processing (e.g. common approach for data editing and imputation).

Many countries have already experienced the trend towards indirect data acquisition in electronic form from administrative registers and distributed databases. Furthermore, direct data gathering through questionnaires and even interviews is increasingly computer-assisted and at least partially done in an EDI form. Finally, new data sources such as electronic commerce records and automated data capturing devices have emerged. The present need to cope with data from different sources raises the difficult question of the efficient combination of multiple data sources. The new techniques of data collection present interesting technical and analytical challenges.

Data processing and analysis: Information technology and statistical techniques play a crucial role in the development of in-house processing of incoming data. Both

nascent and well-developed statistical techniques have become feasible with the computer age. In terms of production efficiency, there is a need to semi-automate the most time-consuming phases, such as data editing and imputation. Thus, there is a need to develop procedures based on techniques such as tree-based classification methods, graphical outlier detection, machine learning such as neural networks or genetic algorithms, etc. There are several specific areas, such as time-series analysis and metadata management, that require both harmonisation and commonly adopted modern methods. An emerging area known as knowledge discovery in databases (KDD) is likely to play a more prominent role in the analysis and data processing efforts of tomorrow.

Dissemination: Official statistics are traditionally presented in the form of printed tables. However, modern electronic forms of dissemination, such as the Internet, are developing rapidly. Thus, there is both a need and a technical possibility to present statistical information in more detail, interactively and using appropriate visualisation tools. The needs expressed by the socio-economic research communities to access individual data to perform their analysis sometimes conflict with the protection of the confidentiality of the data and the respect of the privacy guaranteed by the statistical laws. Improved protection algorithms are needed.



Evolution: Historically, the statistical process was conceived and organised around the concept of the survey. A need was identified, specific concepts and questionnaires were developed, and a survey of the specific statistical units were established. The

data were processed and disseminated, without any special links to other sources of information.

The development of computerised registers (e.g.enterprises, persons), the development of different administrative sources (e.g. tax authorities, employment authorities), the development of automated data collection (e.g. extraction from corporate accounting systems, satellite images, bar codes, the internet) made it possible to conceive a far more integrated approach, where a maximum of data is collected without the active participation of the responding unit. Specific surveys are seen as complementary information sources.

This evolution raises a large number of questions (including ethical questions), but forces the statistical institutes to incorporate new tools and techniques for the collection, integration, and analysis of large databases for safe and efficient dissemination.

3 The European Plan of Research in Official Statistics

Over the last few years, Eurostat has developed, in co-operation with the NSIs, the European Plan of Research in Official Statistics (EPROS), which is an inventory of R&D needs and a guide for further developments. For a complete description see the Web site of the Virtual Institute for Research in Official Statistics (VIROS) at http://europa.eu.int/en/comm/eurostat/research/intro-de.htm.

EPROS is presented following the NORIS classification (Nomenclature on Research in Official Statistics). The first digit classification reads:

1. Methodological issues
2. Advanced technologies for data collection
3. Quality issues
4. Data analysis and statistical modelling
5. Multi-data sources, integration and systematisation
6. Dissemination and disclosure control
7. Other innovative applications of information technologies

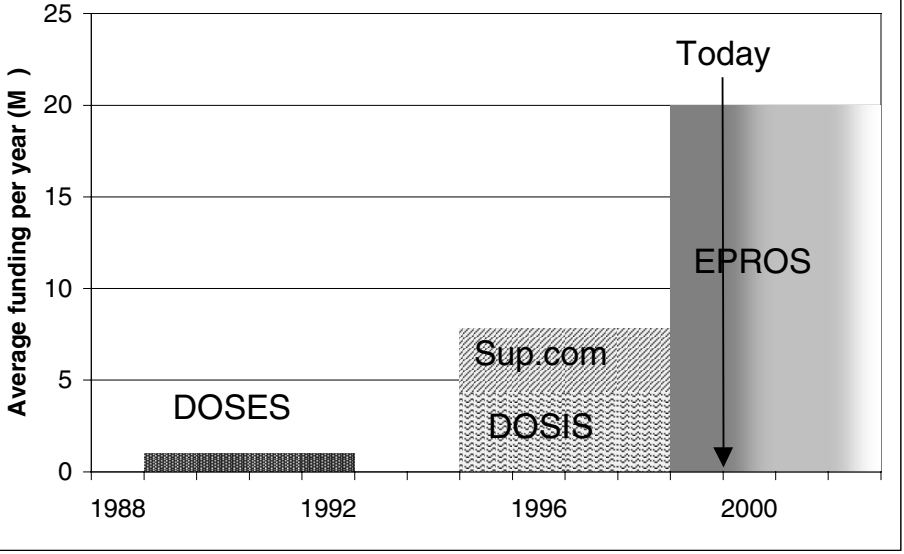
Some of the second digit headings are particularly interesting for DaWaK , such as 2.3 Automatic data capture, 4.1 Data analysis and knowledge extraction, 5.2 Multi-source environments and 5.4 Distributed database management and analysis systems. The reader is referred to the original documents at the above-mentioned Web site for a complete NORIS presentation. Additionally, a number of documents explaining the scope of the various calls for proposals can be found.

4 Research in Statistics through the former EU Framework Programmes

The present paradigm of official statistics is a highly complex production process which must achieve goals of timeliness, accuracy and relevance. Major investment into R&D is required, but the burden is often too heavy for most NSIs. Thus, a critical mass of partners has to be achieved. As most R&D in official statistics are international by definition, the results should be useful both at national and at European level. The role of Eurostat in the co-ordination of research is to achieve the new production model at the national level as well as the European level, which brings a further advantage: that of achieving harmonisation and international comparability..

To face these challenges, Eurostat has promoted the development and the use of new methods and tools since 1989. It has also reinforced the co-operation between the ESS and the scientific communities. The main mechanisms to implement this activity have been the Community R&D Framework Programmes, in which R&D in statistics has steadily grown in importance. Eurostat also has a strong tradition in the organisation of scientific conferences (e.g. KESDA 98, NTTS 98, ETK 99) and Workshops (e.g. Statistical Metadata, Confidentiality), in which areas of particular relevance for Official Statistics are discussed together with academics and representatives of the private sector. Over the last years, the main research programmes supporting the ESS have been the DOSES, SUPCOM and DOSIS programmes. Currently, the EPROS programme is being implemented.

Fig. 3: R&D amounts and periods



SUPCOM: Since 1995, Eurostat has benefited from a specific research budget line of the 4th Framework Programme entitled "Research Support Activities on a competitive basis". This programme was created to fulfil the research needs of the Commission services, and has been used by Eurostat to provide and disseminate existing information and to harmonise the system of data collection and analysis which was in place in the European Union.

Eurostat used this opportunity to launch 93 R&D studies (for a total of 13.7M)on a wide variety of statistical topics, of which 22 were awarded to NSIs. More than two-thirds of these projects have been completed, and the results are progressively made available on the Commission's Website. Overall, half of the SUPCOM projects concentrated on methodological issues, dealing with harmonisation and comparability (often applied to social statistics), environment, transport and business. Another well-covered topic is the adaptation of the statistical system to the new IT realities (e.g. use of the internet, new graphical tools, computer assisted training). Emerging phenomena and techniques were also explored (e-commerce, neural networks, etc.). The first results of an on-going external evaluation demonstrate that 90% of the projects evaluated thus far contribute to the work of Eurostat.

A number of projects fall within the domain of this conference, of which the most interesting are: Data Fusion in Official Statistics, Design of an Integrated Statistical Meta-information System and Creation of a CD-ROM on Metadata for Use in National Statistical Institutes, Databases and Metadata in Reference Systems, and Meta-Databases for Statistical Information. Further information on these and related projects can be found at the VIROS Web site.

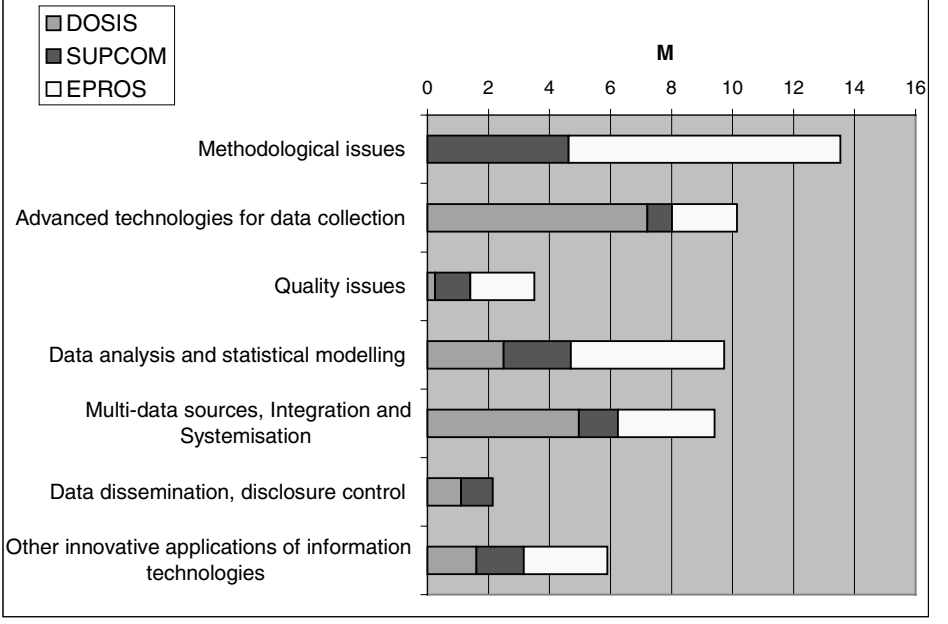
DOSIS: The DOSIS (Development of Statistical Information Systems) projects provide the statistics component of the 4th Framework Programme (1995-1998) with a strong emphasis on the application of results geared to improving the competitive position of European industry and the efficiency of NSIs. It was also intended that the Framework Programme would encourage co-operation through the creation of multinational consortia drawn from academia, government and the private sector. Eighteen projects were co-financed, for a total budget of 17.6 M.

A number of these projects, some of which have already been completed and most of which will come to an end in the course of this year, are within the scope of DaWaK. **ADDSIA** investigates distributed database techniques and the World Wide Web technology to improve access to statistical data for the European research and policy community. **AUTIMP** will develop automatic imputation software for business surveys and population censuses. **TADEQ** will develop a prototype of a tool capable of documenting and analysing the contents and structures of electronic questionnaires. **SDC** addressed the problem of disclosure control of confidential data. Both aggregated data tables, when a unit takes a high proportion of the cell, and microdata, when a record can be identified due to some of its public characteristics, are tackled.

Research in statistical metadata is of particular relevance for the ESS. Several standards exist: international methodologies and classification systems (national

accounts, activity or products nomenclatures, etc.); general guidelines for statistical metadata on the Internet published by UN/ECE; general and special data

Fig . 4: Funding by NORIS and by Programme



dissemination standards from the IMF; and the OECD standard for Main Economic Indicators. Eurostat is participating in this research with the long-term goal to construct an integrated European metadata reference system for statistical data. At the same time, there is a growing need to develop systems for storing and harmonising metadata, without waiting for the various longer term solutions to become operational. Related problems were discussed during the Workshop on Statistical Metadata last February. The following broad priorities were identified: (1) a greater collaboration with international statistical agencies for the harmonisation of metadata content requirements; (2) the setup of networks of experts for developing and implementing specific tools for the management of metadata; and (3) promoting the use of existing metadata standards and tools. Two DOSIS projects have been funded: in order to avoid that the provision of metadata for the user of statistics is a special documentation process that is very time and resource consuming, **IMIM** will develop a metadata management system that is integrated with the whole production process. In addition, **IDARESA** will design and implement an integrated statistical data processing environment making use of meta-information methodology that is aimed at storage, access and dissemination of statistical information.

Furthermore, there are a number of projects dedicated to knowledge extraction and symbolic data analysis in Official Statistics. The goal of the **KESO** project is to develop a prototype knowledge extraction or data mining system for identifying patterns and relationships in statistical datasets. **BAKE** will develop an environment

able to extract Bayesian Belief Networks from databases. The first step will be to investigate the feasibility, relevance and applicability of the approach to official statistics by selecting 10 problems provided by NSIs. Finally, **SODAS** has developed techniques and software to facilitate the use of symbolic data analysis in NSIs. These three projects testify to the strong interest of the ESS in knowledge extraction research. Moreover, Eurostat organised a well-attended conference, KESDA 98, on Knowledge Extraction and Symbolic Data Analysis (see [1] for more information).

Finally, the increased access to large, complex datasets coming from distributed databases necessitates the availability of intelligent visualisation tools that can be tailored to user needs. **IVISS** will develop an interactive visualisation system that can be expanded and customised, and will help the user in the analysis of complex data.

At the time of writing this paper, half of the **DOSIS** projects have been completed.

5 Research in Statistics through the 5th Framework Programme

The fifth Framework Programme, which was initiated 1999, is the first explicitly to recognise the importance of statistics for the information society. Research in statistics was included as a specific action line after consultation with policy makers, NSIs, research communities, and private sector companies that are the users and the developers of new tools, methods and concepts. In response to the challenges that the ESS faces, the EPROS plan has been drafted according to the following principles: that projects should have clear and appreciable R&D content**;** reflect the needs of the ESS and other producers/users of statistics; make use of new developments in technology; have a clear exploitation plan; and rest on cross-national and multidisciplinary partnerships. The implementation of EPROS involves the following groups of actors: official and academic statisticians with theoretical and domain expertise; computer scientists and information managers; other professionals, e.g., economists, in multidisciplinary teams; users, producers and providers of data and information technologies. Five key elements of EPROS are briefly described below:

Research & Development into generic statistical tools, techniques, methodologies and technologies: This covers the whole statistical production process. Hence, it encompasses the definition of concepts and methods through to the analysis and dissemination of statistics, as defined in the NORIS (Nomenclature of Research in Official Statistics) classification. There is particular emphasis on achieving the highest standards of data quality, and on automating data collection, capture, interchange, editing, imputation and dissemination.

Statistical Applications: the aim here is the use of statistical tools, techniques, methods and technologies in support of domain research activities in other parts of the research programmes.

Statistical Indicators for the New Economy (SINE)*:* recent trends in Information and Communications Technology (ICT) together with the evolution of globalisation

have led to the emergence of the **"New Economy".** The long-term structural shift from the industrial economy towards the services and the information economy must be monitored by statistical indicators. It is likely that conventional concepts and classifications will be inappropriate. Hence, there should be considerable emphasis on innovative, forward-looking indicators relating to emerging activities.

Transfer of technology and know-how initiative (TTK)*:* a key objective of the R&D activities is to achieve transfer of technology and know-how in the ESS and other relevant organisations and institutions that could benefit from the knowledge and tools developed within EU research, either steered by the Commission, by national R&D programmes or by NSIs themselves.

Supporting activities: other complementary measures are foreseen to prepare, extend or disseminate the R&D and TTK projects.

In 1999, IST work programme CPA4 (Cross Programme Action 4), two calls for proposals for new indicators and statistical methods were issued. Among others, proposals were invited "to develop and demonstrate new methods and tools for the collection, interchange and harmonisation of statistical data, …". In 2000, under CPA8 (Cross Programme Action 8), foci have been defined, for instance on "statistical data mining, … the use of administrative data for statistical purposes, … improvements in quality and in timely and low-cost data production …".

Thus, three calls for proposals have been issued to date. At the time of writing this summary, only the results of the first call are available and final. Thirteen research proposals of the forty-one proposals submitted to the IST Programme were eventually funded. In the second call, which is dedicated to research on Statistical Indicators of the New Economy (SINE), six proposals have been evaluated positively and the Commission will soon begin negotiating with the consortia. With respect to the third call, which was again open for methods and tools, applications and accompanying measures, some fourteen proposals have proceeded into the second round. A final decision is expected by the end of this year.

The first projects funded under the 5th Framework Programme began in 2000. A large range of research activities is covered: improvement of classifications, automated questionnaires, ex-post harmonisation techniques, editing and imputation, business cycle analysis and metadata (with a specific interest on standardisation). The statistical indicators of the new economy are addressed via the description of the content of the Internet, the tracking of e-commerce, new indicators for the knowledge economy and for innovation through patent databases etc.

From the first call, a number of funded projects is again closely related to the research activities of DaWaK. **EUREDIT** will evaluate and compare the current methods for editing and imputation to establish current best practice methods. In addition, new methods for editing and imputation based on neural networks, fuzzy logic methodology and robust statistical methods will be developed and compared with the

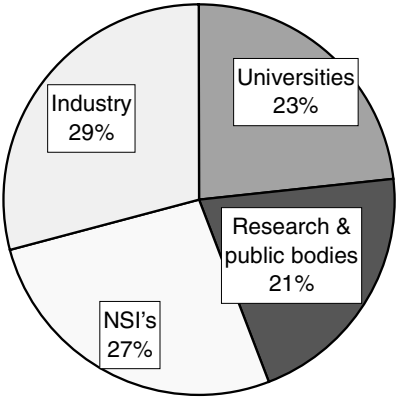
current best practices. **IQML** will provide the solution for the collection of timely statistics to support regional, national and Community policy making, whilst at the same time reducing the reporting burden on enterprises. The technical objective is to understand how to develop an intelligent questionnaire that can interrogate company database metadata and questionnaire metadata in order to extract the required data automatically. **MISSION** will provide a software solution that will utilise the advances in statistical techniques for data harmonisation, the emergence of agent technology, the availability of standards for exchanging metadata and the power of Internet information retrieval tools. The result will be a modular software suite aimed at enabling providers of official statistics to publish data in a unified framework, and allowing users to share methodologies for comparative analysis and harmonisation. The modules will be distributed over the Web and will communicate via agents. **METAWARE** will develop a standard metadata repository for data warehouses and standard interfaces to exchange metadata between data warehouses and the basic statistical production system. The aim is to make statistical data warehouse technologies more user-friendly for public sector users. The project will build upon the results of the 4th Framework Programme (in particular the IMIM and IDARESA projects) and develop object-oriented metadata standards for statistical data warehouses.

6 The future Framework Programmes

As the Framework Programme is implemented through annual work programmes, it is currently too early to detail the activities of the year 2001. However, the main intention for the two next years is to continue the implementation of **EPROS**, whilst taking into account any newly emerging domains.

One initiative that could strongly influence next year's work programme is the recent **eEurope** initiative of the European Union. This initiative was launched by the

Fig. 5: Participants in R&D projects
N=358



European Commission in December 1999 with the objective to bring Europe on-line, by promoting a cheaper, faster and more secure internet, by investing in people and skills, and by stimulating the use of the internet. At the recent Council summit in Lisbon, the Heads of State and Government committed themselves to a number of measures, including a target date to bring eEurope forward. Ten areas were identified for which action on a European level would add value. The ESS should be associated with the collection of the data needed for the benchmarking exercise. Part of the funding for this activity might come from the IST programme. The benchmarking

exercise will be necessary to ensure that actions are carried out efficiently, have the intended impact and achieve the required high profile in all Member States. Furthermore, the next Framework Programme (FP6) is currently under preparation. Eurostat intends to continue the R&D activities in line with the EPROS document. Consultation will take place with the research communities, the NSIs, and the users in order better to understand the needs and technological opportunities for future research.

7 Conclusions

Overall, since 1996, the R&D activity undertaken or supported by Eurostat has been considerable. In the last five years, 55 M have been spent on 130 projects, involving more than 350 research teams covering all fifteen Member States. NSIs from twelve Member States participate in 96 research projects. The aims of promoting research in the ESS, of reinforcing the research potential and of strengthening relations between official and academic statisticians are being gradually achieved.

Eurostat has commissioned an independent evaluation of the research projects funded under the SUPCOM initiative. In the panel discussion of last year's ETK Conference (Exchange of Technology and Know-how) in Prague, it became clear that better mechanisms are needed to transfer the research results into the daily work of the NSIs. Research teams very often develop only prototypes. Effective technology transfer requires further analysis and major adaptations, something that the research teams will not and the NSIs usually cannot deliver. To address these issues, and to co-ordinate the transfer of technology and know-how, Eurostat has set up, jointly with the Joint Research Center in Ispra, the European Statistical Laboratory.

8. References

[1] KESDA '98, Proceedings of the Conference on Knowledge Extraction and Symbolic Data Analysis, European Commission, Eurostat, Luxembourg, 1998.

Elimination of Redundant Views in Multidimensional Aggregates

Nikolaos Kotsis Douglas R. McGregor

Department of Computer Science, University Of Strathclyde
26 Richmond Street, Glasgow, G1 1XH, Scotland
nick,douglas@cs.strath.ac.uk

Abstract. On-line analytical processing provides multidimensional data analysis, through extensive computation based on aggregation, along many dimensions and hierarchies. To accelerate query-response time, pre-computed results are often stored for later retrieval. This adds a prohibitive storage overhead when applied to the whole set of aggregates. In this paper we describe a novel approach which provides the means for the efficient selection, computation and storage of multidimensional aggregates. The approach identifies redundant aggregates, by inspection, thus allowing only distinct aggregates to be computed and stored. We propose extensions to relational theory and also present new algorithms for implementing the approach, providing a solution which is both scalable and low in complexity. The experiments were conducted using real and synthetic datasets and demonstrate that significant savings in computation time and storage space can be achieved when redundant aggregates are eliminated. Savings have also been shown to increase as dimensionality increases. Finally, the implications of this work affect the indexing and maintenance of views and the user interface.

1 Introduction

On-line analytical processing (OLAP) provides multidimensional data analysis to support decision making. OLAP queries require extensive computation, involving aggregations, along many dimensions and hierarchies [Codd93]. The time required to process these queries has prevented the interactive analysis of large databases [CD97]. To accelerate query-response time, pre-computed results are often stored as materialised views for later retrieval. The full materialisation relates to the whole set of possible aggregates, known as the data cube. In this case the additional overhead is orders of magnitude larger than the input dataset and thus very expensive in terms of both the pre-computation time and storage space. The challenge in implementing the data cube has been to select the minimum number of views for materialisation, while retaining fast query response time. In this paper, we propose a new approach which identifies and eliminates redundancy in the data representation of multidimensional aggregates as first described in [Kot00]. The approach identifies redundant views and then computes and stores only the distinct or non-redundant ones. The whole set of aggregates

Y. Kambayashi, M. Mohania, and A M. Tjoa (Eds.): DaWaK 2000, LNCS 1874, pp. 146–161, 2000.
© Springer-Verlag Berlin Heidelberg 2000

can, however, be retrieved later, from the selected subset, in the querying process without any processing cost. The algorithm which identifies the redundant views requires approximately 10% of the time required to compute the data cube conventionally. As the selected subset is smaller than the full data cube, the overall computation is considerably faster than the conventional method described by [GBLP96]. Further contributions of this paper are:

- Novel extensions to relational theory with regard to redundancy in an OLAP context
- New algorithms for the identification and elimination of redundant views
- An extensive set of experimental results confirming the theory by empirical measurements with the goal of demonstrating fairly the practicability of the new approach

The paper is organised as follows: Section 2 describes the fundamental aspects of the OLAP model. Section 3 discusses related work in implementing the data cube. Section 4 describes the Totally-Redundant Views. Section 5 presents the Key-algorithm for the identification of redundant views. Section 6 describes the improved implementation of the data cube. Section 7 presents the results from the experiments conducted in this work. Finally, Section 8 presents the conclusion.

2 The Multidimensional Model

The typical organization of data in a relational OLAP system is based on the star schema design with a *fact table* and several *dimension tables* [Kim96]. For example, a fact table could be the Sales relation with the following schema:

Sales(ProductId, LocationId,TimeID, Sales)
and the following dimension tables:
Product(ProductID, Type, Category)
Location(LocationID, City, Country, Continent)
Time(TimeID, Month, Year)

Each attribute of the Sales relation may be classified either as a dimension or a measure of interest. In this example, the measure of interest is Sales and the dimensions are Product, Location and Time.
The operations involved in OLAP queries are typically aggregates in one or more dimensions. A given measure in n dimensions gives rise to 2^n possible combinations. In the table *Sales* above, the following aggregations are possible: ($Product_Location$), ($Product_Time$), ($Location_Time$), ($Product_ Location_ Time$), ($Product$), ($Location$), ($Time$), (All).
Gray et al.[GBLP96] introduced the Cube-by operator as the n-generalisation of simple aggregate functions. It computes every group-by corresponding to all possible combinations of a list of dimensions. In the previous example the 2^n possible combinations can be expressed by one SQL statement: e.g.

SELECT S.Product S.Location, S.Time as (SUM) Sales
FROM Sales
CUBE-BY S.Product S.Location, S.Time
The benefit of the Cube-By operator is that the user is no longer required to issue all possible group-by statements explicitly. The impact is greater when multiple dimensions with hierarchies are involved, resulting in thousands of explicit Group-by statements. In the next section we will discuss existing methods for the implementation of the cube.

3 Implementing the Data Cube

Several approaches have been proposed for implementing the data cube which differ from the approach proposed in this paper. To the best to our knowledge, there is no work which has considered identification and elimination of redundancy in the data representation of the multidimensional aggregates for the efficient implementation of the data cube. However our approach is compatible with previous approaches and may be integrated with them. The computation of the cube has been studied by [DANR96], [SAG96], [ZDN97], [RS97] and [BR99]. Several optimisations which have been proposed by [GBLP96], [AAD+96] and [SAG96] are: smallest parent, cache results, amortise scans, share sorts and share partitions. In the selection of the materialised views, work has been presented by [HRU96], [BPT97], [Gup97] and [SDN98]. [HRU96] presents the Greeedy algorithm that uses the benefit per unit space of an aggregate. The inputs to the algorithm are the amount of space available for pre-computation and a set initially containing all aggregates in the lattice, except the raw data. The output is the set of aggregates to be materialised. The authors [HRU96] have shown that the benefit of the Greedy algorithm is at least 63% of the optimal algorithm. [SDN98] has evaluated the Greedy Algorithm and shown that it needs a prohibitive amount of processing. The algorithm PBS (Picked by size) is proposed as an improvement of the Greedy algorithm[HRU96]. PBS picks aggregates for pre-computation in increasing order of their size. The algorithm's performance is the same as the greedy algorithm but only requires a fraction of the time needed by the Greedy Algorithm. [BPT97] presents a heuristic for the identification of approximately equally sized group-bys. However their technique is based on size estimation - which is restricted to uniform distributions - and also requires knowledge of functional dependencies. The authors of [BPT97] have also observed that the number of representative queries is extremely small with respect to the total number of elements of the complete data-cube. Thus the indication of the relevant queries is exploited to guide the selection of the candidate views, i.e., the views that, if materialised, may yield a reduction in the total cost. Recently, [BR99] introduced the Iceberg-Cube as a reformulation of the data cube problem to selectively compute only those partitions that satisfy a user-specified aggregate condition defined by a selectivity predicator (HAVING clause). The authors of [BR99] claim that this optimisation improves computation by 40%. However, this approach is inherently limited by the level of aggregation specified

by the selectivity predicate HAVING-COUNT(*). Compressed methods have been also proposed for the materialised views by [OG95], [WB98] and recently [KM99] and approximate methods by,[BS98] and [VW99].

4 Totally-Redundant Views

Totally-Redundant (or *T-R*) views are the views derivable from their parent input relations which are already computed and stored as distinct aggregates. The T-R views are thus virtual representations of their parent input relations and they may be derived from them by a simple projection [1]. Their identification is based on the *g-equivalent* property as explained in definition 4 overleaf. Table 1 introduces notation which will be used to discuss the theory of Totally-Redundant views as follows:

Notation	Description
R	Input (or parent relation of R')
R'	Aggregation (or child relation of R)
t_R	Tuple in R
$t_{R'}$	Tuple in R'
C_R	Cardinality of R
$C_{R'}$	Cardinality of R'
S_t	Set of grouping attributes in t_R
$S_{t'}$	Set of grouping attributes in $t_{R'}$
M_R	Measure of interest in R
$M_{R'}$	Measure of interest in R'

Table 1. Notation of main components

In line with *Codd* [Codd70], when a domain (or combination of domains) of a given relation has values which uniquely identify each element (n-tuple) of that relation it is called a *candidate key.* We classify *candidate keys* into two types: *Definitional* and *Observational keys.*

Definition 1. *Definitional Keys are those keys which are defined as part of the database schema (e.g., by the database designer).*

Definition 2. *Observational Keys are those keys which are defined empirically.*

Thus an Observational key is invariant in a read-only database but may be modified by updates to the dataset. A Definitional key always possess a unique identification property despite updates.

[1] Simple projection refers to a projection which does not require duplicate elimination

Definition 3. *A tuple $t_{R'}$ is defined to be* **group-by - equivalent** *or* **g-equivalent** *($\Rrightarrow$) to a tuple t_R if, and only if, the set of grouping attributes $S_{t'}$ is a subset of the set of grouping attributes S_t and the measure of interest is equal for both R' and R (see Figure 1).*

$$t_{R'} \Rrightarrow t_R \textbf{ iff } S_{t'} \subset S_t \quad \textbf{and} \quad M_{R'} = M_R$$

$$\mathrm{R}'(\mathrm{P2, L1, 30}) \quad \Rrightarrow \quad \mathrm{R\ (P2, L1, T2, 30)}$$

Fig. 1. The g-equivalent tuple

Definition 4. *A relation R' is defined to be* **g-equivalent** *($\Rrightarrow$) to a relation R if, and only if, for every tuple in R' there is a g-equivalent corresponding tuple in R and both relations have the same cardinality.*

$$R \Rrightarrow R' \quad \textbf{iff} \quad (\forall\ t_R\ \exists\ t_{R'} \quad \textbf{such that} \quad t_{R'} \Rrightarrow t_R) \quad \textbf{and} \quad C_{R'} = C_R$$

4.1 Extending the Relational Theory: Totally-Redundant Views

Theorem 1. *When the result relation R' of an aggregation has the same cardinality as the parent relation R then each tuple $t_{R'}$ is g-equivalent to the corresponding tuple t_R, both in its grouping attributes and in its measure of interest.*

Proof In an aggregation operation, each tuple $t_{R'}$ is derived either from a single tuple or from several tuples of the input relation R. If a tuple $t_{R'}$ is a result of several tuples in R then there is a reduction in cardinality of the relation R' relative to the relation R. Thus, if the cardinality of R and R' is the same, then each tuple $t_{R'}$ must have been derived from only a single tuple t_R, and hence must be g-equivalent to the corresponding tuple of R in both its projected dimensional values and its measure of interest.

Theorem 2. *Any aggregation of a relation R over any set of domains which includes a Candidate Key, produces a result relation R' in which each resulting tuple must be g-equivalent to the corresponding tuple of R in both its grouping attributes and its measure of interest.*

Proof Any projection or aggregation of R that includes a candidate key preserves the same number of tuples. Thus, any aggregation or projection which includes a candidate key of R, produces a result relation R' with the same cardinality as R. Thus, (by Theorem 3.1) each tuple in R' must be identical to the corresponding tuple of R in both its projected dimensional values and its measure of interest.

Theorem 3. *(Converse of Theorem 1): When an aggregation or projection of a parent relation R over a set of domains produces a result relation R' with the same cardinality as R, then that set of domains contains an Observational candidate key of both R and R'.*

Proof If an aggregation or projection of a parent relation R produces a resulting relation R' with the same cardinality, then the dimensions over which the aggregation has been carried out must uniquely distinguish each tuple of R. Each resulting tuple must have been derived from a single parent tuple (if this were not so, then aggregation must have occurred, with a resultant reduction in the cardinality of R'). Hence if the cardinalities of R' and R are the same, the dimensions of the aggregation must include a observational candidate key of R.

4.2 Example of Totally-Redundant Views

Figure 2 shows the input relation R and the *g-equivalent* aggregate relation R'. R and R' have equal cardinality and for every tuple in R' there is a *g-equivalent* corresponding tuple in R. The aggregate relation R' is redundant since it can always be produced by a simple projection of R.

R			
Product	Location	Time	Total_Sales
P1	L1	T1	80
P2	L3	T4	20
P1	L2	T3	50
P4	L1	T1	30
P3	L1	T3	80
P4	L3	T2	100
P1	L3	T1	45
P3	L2	T3	70
P2	L1	T2	30

R´		
Product	Location	Total_Sales
P1	L1	80
P2	L3	20
P1	L2	50
P4	L1	30
P3	L1	80
P4	L3	100
P1	L3	45
P3	L2	70
P2	L1	30

Fig. 2. The Input Relation R and its *g-equivalent* aggregation R'

4.3 Utilizing the Totally-Redundant views in the Implementation of the Data cube

Utilising Theorem 3.1 avoids the explicit computation and separate storage of any aggregate which includes a key. This not only has a major impact on the computation time, but also reduces the storage requirements of the materialised views. The T-R approach proceeds in two stages:

- **Stage 1:** Identification of T-R views by determining the set of Observational keys.
- **Stage 2:** Fast Computation of the data cube by avoiding the processing and storage of T-R views.

5 Stage 1: The Key Algorithm

Briefly stated, the approach adopted empirically determines all the Observational keys [2] present in a given parent relation prior to the materialisation of the data cube. The algorithm examines all possible aggregates in the data cube and classifies each either as Totally-Redundant or not. Thus, the algorithm examines whether each group-by includes one of the already detected keys - if so it can be categorised immediately as *g-equivalent* to the input relation and hence Totally-Redundant. The remaining group-bys, with maximum size smaller than the size of the input relation, can not be candidates for the equivalence property. The remaining aggregates are tested - to see whether any two tuples of the input relation are combined during aggregation - using the First Duplicate Detector (FDD) routine described later in this section. The Key algorithm proceeds from the minimum) to the maximum arity and uses the FDD to scan each group-by until it detects the first duplicate tuple. When a duplicate is found, the current group-by is an aggregate (not *g-equivalent* relation) and hence, according to theorem 3.1, not a key. If there are no duplicates, then the schema of the group-by is an Observational Key. With the discovery of a key, the algorithm eliminates from further consideration all subsequent group-bys of greater arity including the key in their schemas. Such group-bys are thus also Totally-Redundant views.

5.1 The Complexity and Performance of the Key Algorithm

Given a relation $R(A1, A2, ..., An)$ with n dimensions, the complexity of the algorithm is $O(C * 2^n)$, where n is the number of dimension attributes and C is the cardinality of each group-by. C is an upper bound since the Key algorithm exits at the first duplicate which is normally detected before completing the scan of the whole group-by. The supersets of each Observational key must also possess the unique identification property and hence these group-bys can also be eliminated by inspection. If a schema of m attributes has been recognized as an Observational key of an *n-dimension* relation R, then the number of times the schema's m attributes will appear in the 2^n possible aggregations is 2^{n-m}. The maximum benefit which can be derived occurs when $m = 1$ and the least benefit when $m = n$ and thus there is no key (n is the superset in R). For example a data cube of ten-dimensions would produce 1,024 aggregates. A key of two dimensions would reject $2^{10-2} = 2^8 = 256$ aggregates as Totally-Redundant and thereby no computation or storage would be required for them. The performance of the Key-algorithm in our experiments required approximately 10% of the conventional data cube processing time.

The Key algorithm
Input: search lattice of the input Relation R

[2] The candidate key can be determined either by definition (database catalogs) or using the [LO78] algorithm to find keys for a given set of attributes' names and a given set of functional dependencies. However, this approach would require knowledge of the functional dependencies.

```
Output: Set of Observational Keys K - array of strings
i := 0;
s := 0;
K := null;
while   i < NoOfCombinations − 1 do  /* NoOfCombinations = 2^n */
    if GroupBy[i].size < R.size then begin   /* The size is an upper bound */
       if GroupBy[i].schema ∈ K(s) then   /* This is a redundant GroupBy */
        i := i + 1;
        else if found duplicate then  /* First Duplicate Detector */
        i := i + 1;  /* This GroupBy is an aggregated relation - it's schema is not a Key */
         else begin
                        s := s + 1;
                        add the GroupBy schema to K(s)
        end;
    end;
i := i + 1;
return set K;
end;
```

The First Duplicate Detector. To examine whether a specific set of domains is an Observation key the First Duplicate Detector (FDD) is used. This is a simplified hash-based aggregation which exits true when a duplicate is found, false if none is detected. The complexity of the algorithm is equal to the cardinality of a relation. In practice however, the average complexity is much smaller since a duplicate tuple is usually found (if it exists) before the full scan of the relation.

5.2 The Recursive Key Algorithm - Derivative Relations

The Key-algorithm, as described in Section 5 identifies *g-equivalent* relations to the input base relation. Further redundancy can be eliminated by applying the Key-algorithm recursively to the derivative aggregate views. The experiments outlined in Section 7.2 indicate that a further reduction in storage of up to 60% can be achieved when the derivatives are utilised. Overall, Totally-Redundant views effect storage savings of up to 85% (TPC-D 60K with 10 dimensions) of the volume.

5.3 Implications of Totally-Redundant views

Potentially the Totally-Redundant views approach can be utilised to reduce the workload in selection, computation, storage, indexing and maintenance of the materialised views in a data warehouse environment. Indexing is not required for the *g-equivalent* views since they can utilise the index of their input parent relation. Maintenance is also faster if no duplication with regard to the *g-equivalent* view is inserted. In this case, no update of this view is needed. Through its identification of redundant views, the Totally-Redundant views approach reduces the browsing time in the interface environment, since views with the same content do not require storage or viewing.

6 Stage 2: Fast Computation of the Data Cube

Although the main algorithm to compute the data cube is based on [GBLP96], our improved implementation utilises the T-R views to avoid computation and storage of aggregates already computed and stored. The improved data cube algorithm has implemented as follows:

```
Input: search lattice of the input Relation R
       Set of Keys K, array of Keys
Output: Set of computed aggregates as views V
begin
i := 0; j := 0; V := null;
    while   NoOfCombinations − 1 > i do  /* From maximum to minimum arity
                                          where NoOfCombinations = 2^n */
    begin
          for j:=0 to entries in K
          begin
               if K(j) ⊂ GroupBy[i].schema  /* Totally redundant view */
               then
                i := i + 1;
               else  /* not a Totally-Redundant view-candidate for computation */
                Aggregate(GroupBy[i]);
                add the GroupBy schema to V[i]
                i := i + 1;
          end;
     end;
    return set V; /* the set of computed views */
end;
```

7 Experimental Evaluation

7.1 Experimental results in Time and Space

We have evaluated our technique by implementing the algorithms described in Section 5 and 6. We compared our approach in time and space with the conventional utilising the smallest parent method as described by [GBLP96]. The proposed algorithms were run on a Dual Pentium 200 MHz with 128 Mb of RAM under windows NT and with 1 GB of virtual memory. No attempt was made to utilize the second processor. Six datasets were used three of which were from the TPC-D [TPC98] benchmark dataset. We have also used two real datasets, the weather data [HWL94] the adult dataset [Koh96] and a small synthetic dataset [Kim96]. From the TPC-D dataset, the lineitem table was chosen at three scale factors 0.1 (600K tuples), 0.01 (60K tuples) and 0.001 (6K tuples). In the TPC-D datasets the measure of interest is the fifth attribute. In all experiments the number of dimensions increases in the order the dimensions appear in the dataset (i.e. where 3 is the selection of the first three dimensions).
The TPC-D 6K dataset. The 6K dataset's attributes' cardinalities were: (1,500),

(200), (10), (7), (50), (3), (2), (2,249), (2,234), (2,241), (4). Figure 3(a) shows the timing results taken from the TPC-D 6K dataset. Although the *T-R* views approach is slower until the five-dimension data cube, after the six-dimension data cube it is faster than the conventional data cube. Figure 3(b) shows the results in space savings where our approach uses 3.8 times less space than the conventional approach. **The TPC-D 60K dataset.** For the 60K dataset, the attributes' cardinalities were: (15,000), (2,000), (100), (7), (50), (35,967), (2,520), (2,464), (2,531), (4), (7). Figure 4(a) presents the time required to compute the data cube by both the *T-R* views approach and the conventional approach. The *T-R* views approach becomes faster after the five-dimension data cube and the increase in performance remains faster in higher dimensions. Figure 4(b) shows space savings of our approach which uses 3.6 times less space than the conventional. **The TPC-D 600K dataset.** The 600K dataset was generated with the scale factor sf=0.1 and cardinalities: (150,000), (20,000), (1,000), (7), (50), (2,526), (2,466), (2,548). For the 600K dataset, the results are similar to the smaller TPC-D dataset. Figure 5(a) shows the timings for our and the conventional approach. After the fifth dimension our approach remains faster than the conventional. Figure 5(b) shows that our approach is approximately 2 times more economical than the conventional approach. **The Hotel dataset.** This dataset is taken from a business example [Kim96]. It is a synthetic dataset with a fact table of Hotel Stays schema with eight dimensions and three measure attributes. The size of the dataset is 2,249 tuples and it was selected to ensure that our performance results in time are not effected by thrashing. The dataset and all its derivative aggregates, even in the conventional approach, almost fitted into the main memory. The cardinalities of the dataset were: (183), (26), (100), (20), (20), (2,168), (2), (20), (4), (907), (366) and (10). In time aspects Figure 6(a) shows that beyond the fifth dimension the performance of the *T-R* approach increases. Figure 6(b) shows that the *g-equivalent* representation is 7.6 smaller than the conventional - as a result of the high number of Observational Keys which have been identified from the Key Algorithm. **The Cloud dataset.** We used the real weather data on cloud coverage [HWL94]. The dataset contains (116,635) tuples with 20 dimensions. The following 10 cardinalities were chosen : (612), (2), (1,425), (3,599), (5), (1), (101), (9), (24). The measure attribute was the sixth dimension. The time results in Figure 7(a) show that the *T-R* approach is not faster than the conventional approach. The reason for this is the high skewness of the dataset which resulted in the non-existence of Observational keys in the input base relation. Also the results in Figure 7(b) show that the space used by our approach is 1.13 times less than the conventional approach. This is lower than in previous cases using TPC-D datasets. However, Observational keys have been found in this dataset, in the derivative relations and the savings have been shown to be much greater than 1.13 (see Figure 11). **The Adult Database.** This is a real dataset from the US Census Bureau in 1994 which was first presented by [Koh96]. Six of the attributes are numerical and the remaining six are categorical attributes. The down-loaded version and was projected to a relation with nine attributes, with the 9th attribute defined as the measured attribute. The cardinalities of the new dataset were: (72), (10), (19,988), (16), (16),(16), (7), (16), (7), (1). The last attribute was used for measure of interest. Figure 8(a) compares the timings from both the *T-R* and the conventional approaches. The time of the *T-R* approach is 60% slower than the con-

ventional approach. The skewness of the data distribution causes the non-existence of Observational keys in the base relation. This is confirmed by the space savings results shown in Figure 8(b), which are not as great as those in the synthetic datasets. In the next section, we show that savings increase when the Observational key is searched in the derivative relations (see Figure 11(b)).

7.2 Totally-Redundant Views of Derivative Relations

T-R views can be found in aggregates other than the base relation. The experiments conducted compare the savings in space when the base relation is used as a reference to the savings achieved when the derivative relations are used as references. On average, for a base relation with ten dimensions, when utilising all derivative relations, the total savings increase to, on average, 90% of the total views in the data cube. This is in contrast to an average of 74% of total views achieved when the base relation was used. For each of the six datasets, the tests were run in several dimensions with the number varying from three to twelve. The savings in space and time after the elimination of the *T-R* views are greater for relations with high dimensionality. Figure 9(a) compares the volume of the *T-R* views, for the 600K dataset, when they are *g-equivalent* to the base relation with the volume of the *T-R* views when they are *g-equivalent* to the derivatives. For the 600K dataset in seven dimensions, the results show a further redundancy of 23.5% compared to those found in the base relation. Figure 9(b) illustrates the results of experiments using the TPC-D 60K dataset. This shows that redundancy when the derivative aggregates have been searched is approximately 15% more than the redundancy found by utilising only the base relation. Figure 10(a) compares the same quantities for the 6K dataset. The redundancy found in this dataset is approximately 21% more than that found by using the base relation as a reference for the g-equivalence property. For the hotel dataset, the redundancy in ten dimensions was 13.3% more for the derivative relations method than that of the base relation. These results are illustrated in Figure 10(b). The Key-algorithm when applied to the derivative relations is particularly effective in the real datasets. Results for the weather dataset reveal that redundancy is 48.41% more than that found by the simple key algorithm. The simple Key-algorithm had identified only 10.26% of the data cube redundancy, compared to tests run in the derivative relations, in which the redundancy was 59.93%. Figure 11(a) shows the results of six different data cube trials, for the weather dataset. Figure 11(b) shows results for the adult dataset. For the simple Key-algorithm, redundancy of *T-R* views (based on the base relation) was 1.88%, in contrast to the redundancy identified in derivative relations of 65.28%. Savings in storage and time for the *T-R* approach vary according to the distribution and sparsity of the trial datasets. Experiments in real datasets, utilising the simple Key algorithm, show that the performance of *T-R* approach was negatively affected by the skewness of the datasets. The aim of the following experiments was to identify the redundancy in derivative aggregates and show that applying the Key-algorithm recursively can lead to substantial increases in space savings. The performance in time is strongly related to the savings in space, since eliminating redundant views results in less computational effort. The experiments conducted in this section revealed that the performance of *T-R* views is affected less by the data skewness when the derivative relations are utilised as parents.

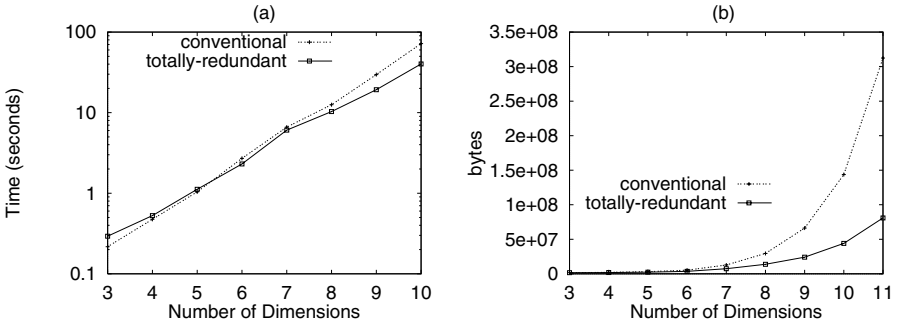


Fig. 3. The TPC-D 6K dataset

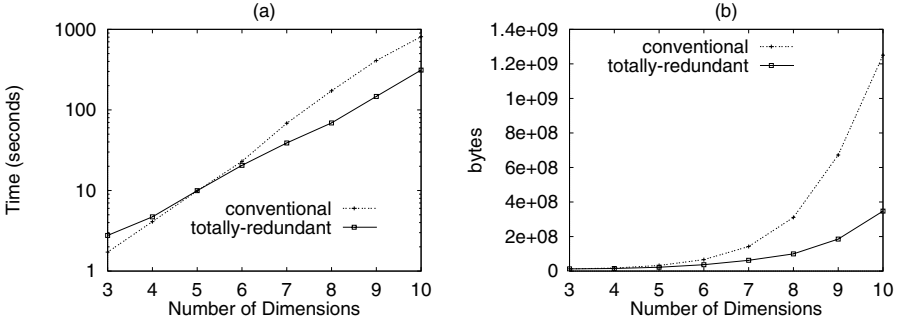


Fig. 4. The TPC-D 60K dataset

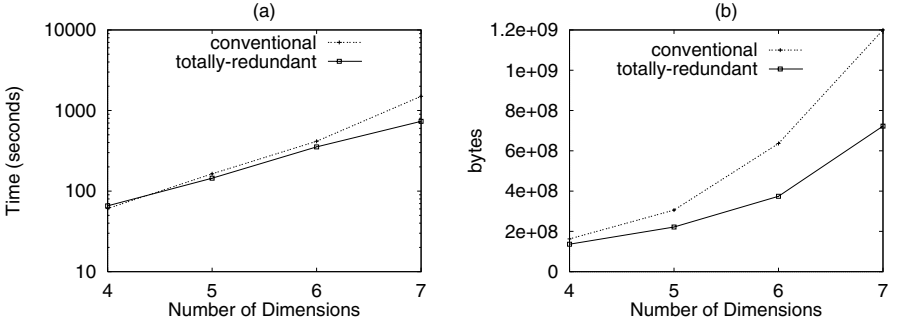


Fig. 5. The TPC-D 600K dataset

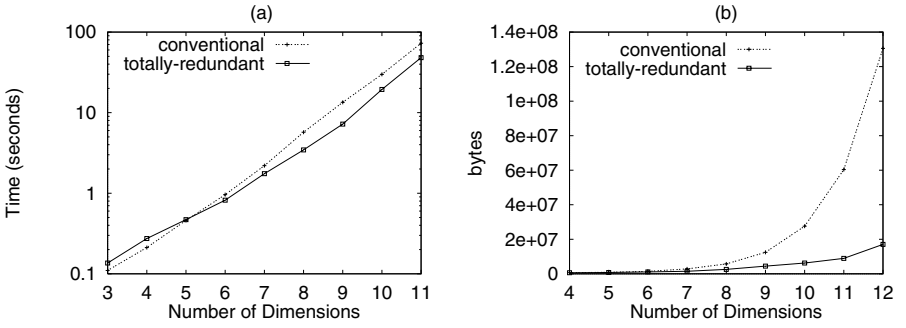


Fig. 6. The Hotel dataset

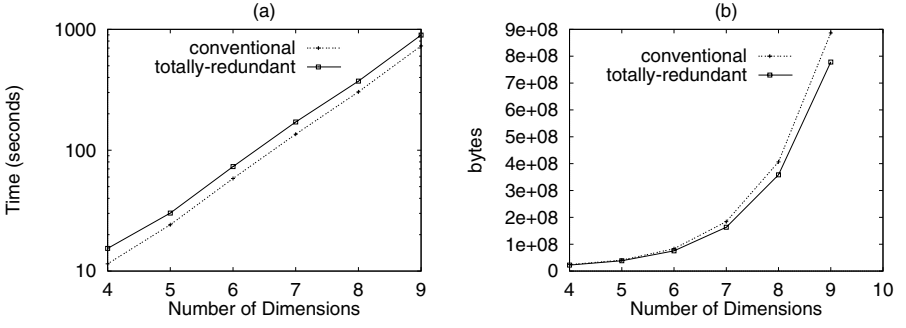


Fig. 7. The Cloud dataset

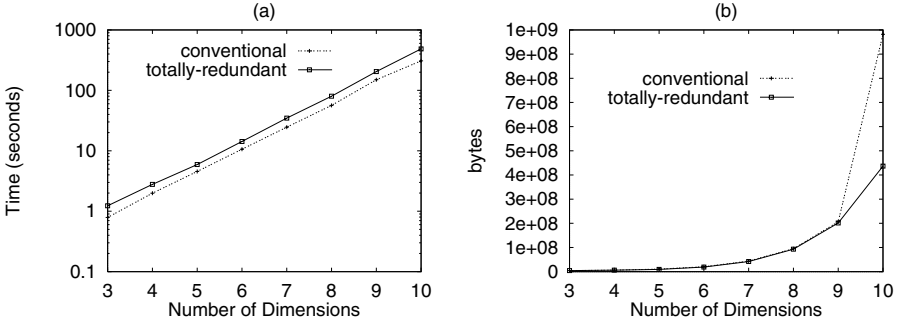


Fig. 8. The Adult dataset

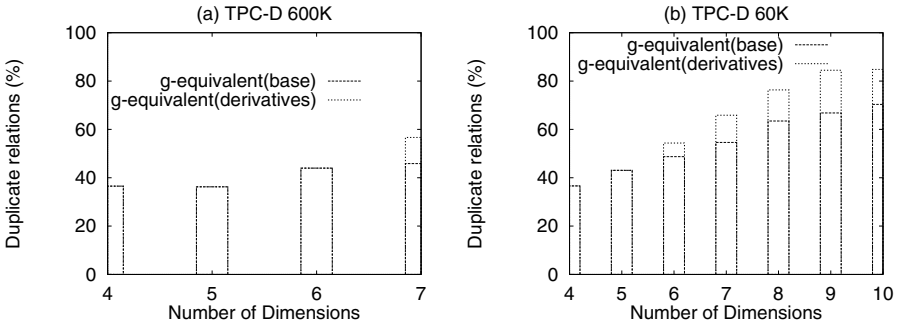


Fig. 9. The effect of Totally-Redundant views in space for the TPC-D Lineitem Table 600K (a) and 60K (b) datasets

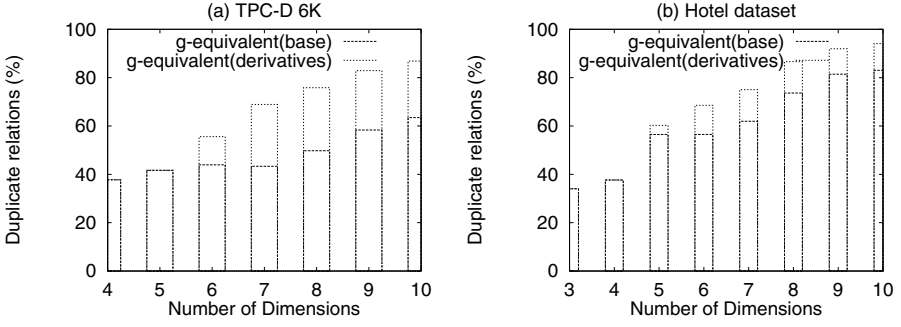


Fig. 10. The effect of Totally-Redundant views in space for the TPC-D Lineitem Table (6K)(a) and Hotel datasets (b)

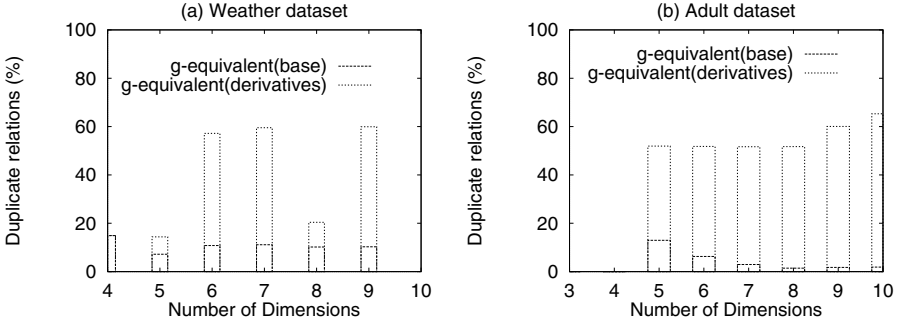


Fig. 11. The effect of Totally-Redundant views in space for the Cloud (a) and Adult datasets (b)

8 Conclusion

We have shown that the identification of redundancy in the data representation of multidimensional aggregates is an important element in the efficient implementation of the data cube. The cost of implementing the whole data cube, when the Totally-Redundant views approach is utilised, is the cost of computing and storing the non-redundant views. Thus, the larger the percentage of redundant views in the data cube, the faster the computation compared to the conventional approach. After the identification of non-redundant views, the whole data cube is materialised without additional cost. We propose extensions to relational theory to identify by inspection those views which are g-equivalent to their parent input relation. The proposed algorithms were evaluated under experimental results. The experiments demonstrate the significance of our approach with regard to the reduction of storage costs without a significant compromise in time. The data skewness negatively affects the existence of Observational keys in the base relation. However, when the derivative relations are used as parent relations, the data skewness does not affect the existence of keys, and the savings increase significantly. Potentially savings in space mean that a smaller number of views are materialised. This results in less computational effort. Furthermore, we discussed the implications of our approach in indexing, maintenance and the user interface. In future work, we will examine the effects of the redundancy in the above areas more thoroughly.

References

[AAD+96] S. Agarwal, R. Agrawal, P.M. Deshpande, A. Gupta, J.F. Naughton, R. Ramakrishnan, and S. Sarawagi. On the Computation of Multidimensional Aggregates. *In Proceedings of the 22nd International Conference on Very Large Databases*, pages 506-521, Mumbai, Sept. 1996.

[BPT97] E.Baralis, S. Paraboschi, E. Teniente. Materialized View Selection in a Multidimensional Database. *Proceedings of the 23rd International Conference on Very Large Databases*, pages 156-165, Athens 1997.

[BS98] D. Barbara, M. Sullivan. Quasi-Cubes: A space-efficient way to support approximate multidimensional databases. *Technical Report-Dept of Inf and Soft.Eng.* George Mason University 1998.

[BR99] K., Beyer, R., Ramakrishnan Bottom-Up Computation and Iceberg CUBEs. *Proc. ACM SIGMOD International. Conf. on Management of Data 1999*, pages 359-370, Philadelphia PA, USA, June 1999.

[Codd70] E.F., Codd, "A relational model for large shared data banks. *Communications of the ACM*, 13(6):377-387, 1970.

[Codd93] E.F., Codd, S.B. Codd, C.T. Salley. Providing OLAP (On-Line Analytical Processing) to User Analyst: An IT Mandate. *Arbor Software* at http://www.arborsoft.com/OLAP.html.

[CD97] S. Chaudhuri, U. Dayal. An Overview of Data Warehousing and OLAP Technology. *Technical Report MSR-TR-97-14, Microsoft Research* Advanced Technology, March 1997.

[DANR96] P.M. Despande, A. Shukla, J.F. Naughton, K.Ramaswamy. Storage Estimation of the Multidimensional Aggregates. *In Proceedings of the 22nd International Conference on Very Large Databases*, pages 522-531, Mumbai, Sept. 1996.

[GBLP96] J. Gray, A. Bosworth, A. Layman, and H. Pirahesh. Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals. *Proceedings of the 21st International Conference on Data Engineering*. New Orleans (LA), USA, Feb. 26-March 1), 1996.

[Gupt97] H.Gupta. Selection of Views to Materialize in a Data Warehouse. *Proc. of the 6th International Conference in Database Theory (ICDT)*, pages 98-112, Delphi, Jan 1997.

[HRU96] V. Harinarayan, A. Rajaraman, J.D. Ullman. Implementing Data Cubes Efficiently. *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 205-227, 1996.

[HWL94] C.J. Hahn, S.G. Warren, J. London. Edited synoptic cloud reports over the globe 1982-1991. Available from http://cdiac.esd.ornl.gov/cdiac/ndps/ndp026b.html, 1994.

[Kim96] R. Kimball. The Data Warehouse Toolkit. *John Wiley 1996.*

[KM99] N. Kotsis, D.R. McGregor. Compact Representation: An Efficient Implementation for the Data Warehouse Architecture. *1st International Conference, DaWAK '99* pages 78-85. Florence, Italy, August 99.

[Kot00] N. Kotsis. Multidimensional Aggregation in OLAP systems. PhD thesis. Department of Computer Science, University of Strathclyde, February 2000.

[LO78] C. Lucchesi, S.Osborn. Candidate keys for relations. *J. Computer and System Science*, 17(2:270-279, 1978.

[OG95] O'Neil, P., Graefe, G., Multi-Table Joins through Bitmapped Join Indices. *Proceedings of ACM SIGMOD International Conference on Management of Data*, 1996.

[RS97] K.A. Ross, D.Srivastava Fast Computation of Sparse Datacubes. *Proc. of the 23rd International Conference on Very Large Databases*, pages 116-125, Athens 1997.

[SAG96] S. Sarawagi, R. Agrawal, A. Gupta. On Computing the Data Cube. it Research report 10026, IBM Almaden Research Center, San Jose, California, 1996.

[SDN98] A. Shukla, P.M. Despande, J.F. Naughton. Materialized View Selection for Multidimensional Datasets, *Proc. of 24rd International Conference on Very Large Databases*, pages 488-499, New York 1998.

[TPC98] F. Raab, editor. TPC Benchmark TM-Standard Specification Revision 1.3.1 Transaction Processing Council 1998

[VW99] J.S. Vitter and M. Wang. Approximate Computation of Multidimensional Aggregates of Sparse Data Using Wavelets. *Proc. ACM SIGMOD International Conference on Management of Data* 1999, p.193-204, Philadelphia PA, USA, June 1999.

[Wid95] J. Widom. Research Problems in Data Warehousing. *Proc. of the 4th Intl. Conference of CIKM*, pages 25-30, Nov.1995.

[WB98] M.C. Wu and A. Buchmann. Encoded bitmap indexing for data Warehouses. *International Conference on data Engineering*, pages 220-230, 1998.

[ZDN97] Y. Zhao, P.M. Deshpande, J.F. Naughton. An Array based Algorithm for Simultaneous Multidimensional Aggregates, *Proceedings of ACM SIGMOD International Conference On Management of Data*, pages 159-170,1997.

Data Cube Compression with QuantiCubes

Pedro Furtado and Henrique Madeira
CISUC/DEI

Abstract: Data warehouses typically store a multidimensional fact representation of the data that can be used in any type of analysis. Many applications materialize data cubes as multidimensional arrays for fast, direct and random access to values. Those data cubes are used for exploration, with operations such as roll-up, drill-down, slice and dice. The data cubes can become very large, increasing the amount of I/O significantly due to the need to retrieve a large number of cube chunks. The large cost associated with I/O leads to degraded performance. The data cube can be compressed, but traditional compression techniques do not render it queriable, as they compress and decompress reasonably large blocks and have large costs associated with the decompression and indirect access. For this reason they are mostly used for archiving. This paper uses the QuantiCubes compression strategy that replaces the data cube by a smaller representation while maintaining full queriability and random access to values. The results show that the technique produces large improvement in performance.

1. Introduction

The amount of information stored and analyzed in data warehouses can be very large. Data cubes are frequently used to materialize views of data sets for Online Analytical Processing (OLAP). They can be very large and must be stored and retrieved from disk in costly I/O operations.

The multidimensional data cube representation uses fixed-size cells and has the advantages of full queriability and random access, as the position of any value is directly computable from the offset on each dimension axis. The major drawback of this representation is that it can occupy a lot of space, causing large access overheads. This has lead us to investigate compressed representations of normal data cubes that would reduce the number of I/O operations needed without obstructing queriability and random access.

Compressing the data cube has important advantages. The most obvious one is that it saves space (between 50% and 80% when QuantiCubes is used). However, the largest advantages are consequences of the smaller space usage. Whenever the memory is scarce compared to the size of the data sets composing the workload, those savings can mean less disk swapping, as more data fits in memory buffers and cubic caches. An even more important effect is that the most time consuming queries over data cube chunks stored on disk become almost 2 to 5 times faster simply because only one half to one fifth of the data has to be loaded from disk (chunk storage organization of data cubes on disk is discussed in [6]).

Traditional compression techniques are not useful in this context. They achieve large compression rates that are very useful for archiving (e.g. Arithmetic Coding, Lempel-Ziv, Huffman Coding, …) [2,5,7,8]. However, the compressed data sets are not directly queriable without prior decompression of whole blocks, due to the variable-length of the compressed fields. The term "variable-length field" refers

Y. Kambayashi, M. Mohania, and A M. Tjoa (Eds.): DaWaK 2000, LNCS 1874, pp. 162–167, 2000.
© Springer-Verlag Berlin Heidelberg 2000

to the usage of a variable number of bits to represent compressed values. Compression/decompression is done by blocks and require expensive access methods to locate the correct block. This indirect access feature accounts for a large overhead. The decompression overhead of most typical techniques is also significant, as it includes bit processing operations and access to auxiliary structures such as dictionaries or code trees to decode the bitstream.

In this paper we present the QuantiCubes technique, which packs reduced fixed-width representations of the individual cells to compress the data cube as much as possible for faster analysis, while maintaining the full queriability and random access properties of the uncompressed data cube. These results are only possible because QuantiCubes guarantees insignificant decompression overhead and maintains the same degree of random access and queriability of the uncompressed data cube, because it stores fixed-width compressed values. The technique is presented and tested in this paper.

The paper is organized as follows: section 2 presents the QuantiCubes technique. Section 3 presents experimental results and section 4 concludes the paper.

2. The QuantiCubes Technique

QuantiCubes transforms the normal data cube representation. The data is analyzed first to determine a fixed-width compressed coding that is optimal with respect to a set of error and space criteria. This is achieved using an adaptable quantization procedure. The data cube values are then coded and packed to achieve a very large compression rate. For example, a data cube cell occupying 4 bytes is compressed to the size of the bitcode that was determined for that cell (e.g. 9 bits). Finally, extremely fast decompression of individual values "on-the-fly" is achieved by unpacking and dequantizing the values with very fast, low-level operations.

2.1. Data Analysis

Quantization [4] is used in QuantiCubes to determine the set of representative values for an attribute. The crucial issue of accuracy is guaranteed by our new proposal for an "adaptable" quantization procedure. This adaptable procedure uses an accuracy constraint based in user-defined or preset error bounds during the determination of quantization levels. This way, the number of levels that are necessary is determined adaptively and attributes that cannot be estimated with large accuracy using a reasonable number of quantization levels can be left uncompressed. The new adaptable quantization algorithm is not discussed here for lack of space. A complete discussion of the algorithm is available in [1].

The specific OLAP context is important in the adaptable quantization procedure, because many fact attributes measuring quantities such as daily or periodic sales offer favorable cases for the replacement by a number of representative values, as they are often distributed around clusters and repeated values. The best results (most accurate and with less bits) are achieved when the data set is made of a small

set of distinct values that are repeated all the time. In that case the quantization levels coincide with distinct values and the bitcodes are simply a more compact representation of the values. In the extreme case, an attribute could have a small number of distinct values in a very large number of rows (e.g. 10M rows). Quantization also returns very good approximations when a significant fraction of the values are distributed around sets of clusters, in which case reconstruction values converge to the centers of distinct clusters. As more levels are added, the quantizer finds the most appropriate positions for the quantization levels to represent most values accurately.

Both the analysis and experimental results in [1] show that the approach guarantees accuracy and typically provides almost exact value reconstruction and precise query answers. This is an important issue, as the technique maintains the queriability of uncompressed data but is also able to guarantee high levels of accuracy for every query pattern and even in the reconstruction of individual values.

2.2. Reconstruction Array

As with many other compression techniques, an auxiliary structure must be accessed to convert between values and bitcodes. However, a small reconstruction array is used and a single direct access to a cell is required for decompression. Figure 1 shows an example that illustrates a reconstruction list. It is a simple array ordered by the reconstruction values with typical sizes between 64 and 8192 values, corresponding to bitcodes with 6 to 13 bits. In this list, bitcodes coincide with the index of the cell. The array is accessed to retrieve the bitcode (index in the array) or the reconstructed value (the value in each position of the array).

3.234	9.21	14.764	21.37	…	102.54

Figure 1 – Illustration of a Reconstruction List

During compression, the reconstruction array is accessed to translate values into the bitcodes of the reconstruction values that are nearest to those values (the bitcodes coincide with the indices, as mentioned above). During decompression, the bitcodes are translated back to the corresponding reconstruction values.

2.3. Compression

Compression requires an access to the reconstruction array to retrieve the bitcode that replaces the value and a concatenation of that value into a word. Each word contains a small number of packed bitcodes and is sized to minimize the overhead of decompression [1].

The lookup in the reconstruction array requires a binary search of the array to determine the quantization level. The search must find the reconstruction value that is closest to the value that must be compressed and return the corresponding index. This

bitcode is then packed (concatenated) into the word using two low-level operations (shift = <<, OR=|, AND=&): $bitcode_i$ = ((data_register <<$shift_i$) | $maskregister_i$) ;

2.4. Decompression

The most crucial operation is decompression, as it is used for "on-the-fly" reconstruction of values during the processing of queries. It requires unpacking the bitcodes and a direct access to the reconstruction array, which is conveniently small to fit in the microprocessor internal cache after a few initial misses. Unpacking uses the operation: $bitcode_i$ = ((data_register >>$shift_i$) & $maskregister_i$);
The reconstruction value is retrieved using a direct access to the reconstruction list:

Reconstructed_value = Reconstruction_List[bitcode];

With this strategy, the unpacking operations are extremely fast.

3. Analysis and Evaluation of the Technique

We have measured 2 to 5 times improvement in access to the compressed data cube on disk, due to the much smaller number of accesses that is necessary to load the compressed cube, while also saving a large amount of space. This is shown in Figure 2, which compares the time taken to load a normal and a compressed data cube, when the compressed data cube has ¼ of the size of the original data cube, as shown in the first column.

Size(MB) Normal/Cmprss	Cmprss	UnCmprss
100/25	11.837	49.431
200/50	23.434	104.01
400/100	48.54	212.158
600/150	73.814	333.631
800/200	112.99	495.07

Figure 2. Loading Time (secs)

In fact, the data cube does not need to be decompressed before operation or analysis even when in memory, as online decompression of individual values does not degrade the performance. This property saves a significant amount of memory buffer space. Assuming that Y values are operated upon in the OLAP query (we use the example Y =100 million values with 4 bytes each – 400MB - and 4 times compression rate), the following operations are necessary:

Uncompressed data cube:

4Y bytes loaded from memory (100 M values = **400 MB**)

+ Y × aggregation-related operations (100 M operations)

Compressed data cube:

Y bytes loaded from memory (100 M values = **100 MB**)

+ Y unpack operations (100 M ops) + M lookup operations (100 M ops)
+ Y × aggregation-related operations (100 M ops)

This discussion shows that operation on the compressed cube accesses a much smaller number of memory words, but the bitcodes must be unpacked and looked up. Predicting which of these operations is fastest is not trivial. It depends on the scheduling of instructions, the time taken to execute them and the use of internal caches by the microprocessor in the lookup functionality. However, by compiling and executing the code in two different systems, it was possible for us to conclude that the overhead incurred by the decompression is not larger than the overhead required to fetch the excess of uncompressed values. Figure 3 shows the time taken to operate on the compressed and uncompressed data cubes and also on an optimized version of the uncompressed data cube that uses loop unrolling [3] to try to match the speed of operation on the compressed data cube. This code was executed in a Pentium II-300Mhz processor.

Size(MB) Normal/Cmprss	Cmprss	UnCmprss	UnCmprss, Unrolled
25/6.25	0.601	1.352	0.610
50/12.5	1.132	2.744	1.192
100/25	2.353	5.498	2.354
150/37.5	3.455	8.232	3.555
200/50	4.607	11.026	4.736

Figure 3 – CPU Execution Time in Sum Operation

The adaptable quantization procedure produces an approximate data set. However, our new proposal for adaptable quantization allows the algorithm to find the appropriate number of levels that are necessary. Figure 4 shows the relative error (average, maximum and standard deviation) of the reconstuction of individual values taken by the attribute "Account Balance" of a bank data set as more bits are added to the quantizer (each additional bit doubles the number of levels available). This attribute had irregular distribution. The result shows that the strategy is able to adapt the number of levels until the error constraints are met. In this case, the quantity was well estimated using 10 to 12 bits.

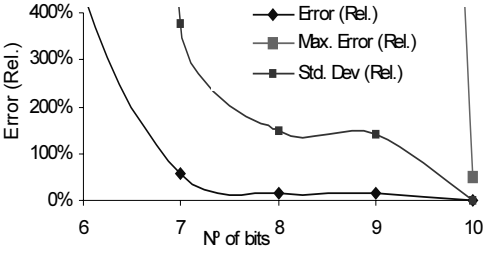


Figure 4-Adaptable Quantization of Attribute Account_Balance

Our experimental evaluation included other experiments that are not shown for lack of space. A more complete discussion and set of experiments is included in [1]. Those results include the comparison of decompression overhead with other compression techniques, showing that QuantiCubes is at least 10 times faster. The compression overhead was also tested experimentally. Although compression is slower than decompression, we have determined experimentally that the time taken to compress and store the data with QuantiCubes is not larger than the time taken to store the uncompressed data, because the compressed data cube is much smaller.

Conclusions and Future Work

In this paper we have proposed a technique (QuantiCubes) which is able to compress data cubes 2 to 5 times and still maintain complete queriability and random access. We have shown that the decompression overhead is completely insignificant, so that "on-the-fly" decompression of any number of values is possible in any situation without significant degradation of performance. The technique is also able to deliver 2 to 5 times faster operation than the uncompressed data. The fast operation of the technique is analyzed in detail and confirmed using experimental results. The technique is used in the context of analysis and operation on the data cube. It is not intended to replace data compression techniques in their archiving or longer-term disk storage function, as they achieve very large compression rates using a variable-width, exact representation of the data set.

References

1. Furtado, Pedro "Accurate Reduced Representations of Multidimensional Fact-Like Data Sets", PhD thesis, Universidade de Coimbra, 2000.
2. J-L Gaily and M. Adler. Zlib home page. http://quest.jpl.nasa.gov/zlib/.
3. J. Hennessy and D. Patterson, "Computer Architecture: A Quantitative Approach". Morgan Kaufmann, 1990: ISBN 1-55860-069-8.
4. S.P. Lloyd. Least Squares Quantization in PCM. IEEE Transactions on Information Theory, IT-28:127-135, March, 1982.
5. M. Nelson, J-L Gaily, "The Data Compression Book", 2nd edition, 1996 - M&T Books, ISBN 1-55851-434-1.
6. S. Sarawagi and M. Stonebraker, "Efficient Organization of Large Multidimensional Arrays", in Proceedings of the 11th Annual IEEE Conference on Data Engineering (ICDE'94) , Houston, Texas, 1994.
7. Welsh, Terry, "A Technique for High-Performance Data Compression" IEEE Computer, Volume 17, Nº 6, June 1984, pages 8-19.
8. J. Ziv, Lempel, "A Universal algorithm for sequential data compression", IEEE Transactions on Information Theory, Volume 23, Nº 3, May 1977, pages 337-343.

History-Driven View Synchronization

Andreas Koeller and Elke A. Rundensteiner*

Department of Computer Science
Worcester Polytechnic Institute
100 Institute Road. Worcester, MA 01609–2280
{koeller|rundenst}@cs.wpi.edu

Abstract. Views over distributed information sources rely on the stability of the schemas of these underlying sources. In the event of meta data changes in the sources, such as the deletion of a table or column, such views may become undefined. Using meta data about information redundancy and user preferences, views can be evolved as necessary to be defined on a modified information space after a source meta data change, while assuring the best possible compliance to preferences that users may have.
Previous work in view synchronization focused only on deletions of schema elements. We now offer the first approach that makes use of additions also. The algorithm is based partly on returning view definitions to previous versions by "backtracking" in the history of views and meta data. This technology enables us to adapt views to temporary meta data changes by cancelling out opposite changes and allows undo/redo-operations on meta data without deteriorating the quality of the view. The mechanism described in this paper will therefore improve the quality of evolved views.

1 Introduction

In recent years, the number of digital information storage and retrieval systems has increased immensely. Many of those database systems are available in some kind of network, and the task of integrating data from different source databases is an increasingly important one. Solutions to this problem include data warehouses, a term which in the database community refers to materialized views over distributed information sources. Newer developments, such as the ubiquity of the World Wide Web, emphasize the independence of data producers from data consumers. Independent data producers or providers have control over the schema of their information sources which raises the question of the influence of *meta data updates* (deletions of attributes or relations in underlying databases)

* This work was supported in part by several grants from NSF, namely, the NSF NYI grant #IRI 97–96264, the NSF CISE Instrumentation grant #IRIS 97–29878, and the NSF grant #IIS 97–32897. Dr. Rundensteiner would like to thank our industrial sponsors, in particular, IBM for the IBM partnership award.

Y. Kambayashi, M. Mohania, and A M. Tjoa (Eds.): DaWaK 2000, LNCS 1874, pp. 168–177, 2000.
© Springer-Verlag Berlin Heidelberg 2000

on a view [RKZ00]. In traditional views (as introduced in the literature handling data updates, e.g., [BLT86, AASY97]), meta data updates can render a view definition undefined.

Some work has been done in the *EVE*-Project [RLN97, NLR98, KRH98] that deals with the rewriting of views under schema updates in the underlying sources in the case of *schema element deletions*, retaining all or a part of the view extent in the new rewritten view. We described algorithms allowing generated view rewritings to be non-equivalent to the original query [LNR97], and presented a model for a numeric assessment of the quality and cost of such rewritings [LKNR99], allowing for the automatic selection of the best view rewriting among many generated ones.

One drawback of the previous work mentioned is the finality of a view synchronization operation. After the deletion of an underlying relation that has been used by a view, the view is rewritten to not refer to that relation any more. Even if the same relation is later added back to the information space (for instance, after a temporary unavailability due to a network problem), it will never be used by the view again since without a global domain model the view synchronization algorithm cannot determine in what relationship a new data element stands to other previously available elements.

Note that with all view synchronization algorithms that have been devised so far, only one-step view synchronization is possible (cf. Section 3). The existing algorithms POC/SPOC [NR98], SVS [RLN97], CVS [NLR98], and GRASP [NR98] cannot synchronize a view under a sequence of schema changes. Rather, they generate a new view definition after each schema change in an information source, without taking any information about earlier IS changes into account. Also, they do not support any meaningful synchronization under *add*−capability-changes. This work describes a way to overcome this shortcoming of previous approaches.

In this paper, we present a complete approach capable of handling all information source schema changes, namely *adds*, *renames*, and *deletes* of attributes and relations. The main contribution of this paper is the utilization of additional available meta data to keep views as close to their original definition as possible, under a sequence of meta data changes that occurs over time.

2 Related Work

Materialized views over distributed information sources have been explored for a number of years. First work focused on questions of materialized view maintenance under data updates in the sources [BLT86, GM95]. Work on maintenance of the more recent mediator-based (constrained) heterogeneous database systems was done by Lu et al. [LMSS95]. More recently, questions of optimizing view queries given varying parameters or capabilities of underlying sources have also been explored [AASY97, GMHI+95]. Generally, work in this area assumes that the rewritten view query computes a view extent *equivalent* to the original one. Prominent approaches that deal with equivalent query rewriting include work by Jarke et al. [JK84] or Agrawal et al. [AASY97]. Important work on inte-

grating heterogeneous sources in one view using a common semistructured data model (OEM) has been done in the TSIMMIS-project [GMHI+95]. Chaudhuri and Shim [CS96] present work on query optimization under user-defined SQL predicates (e.g., stored procedures), which deals with finding optimized query plans for such queries but does not consider changing or non-equivalent queries. Finkelstein [Fin82] describes a model to achieve query optimization in an environment in which a sequence of related (similar) queries is executed on a query engine (using *query-graphs*).

Work on rewriting queries using views (e.g., [LMSS93]) is used in subsequent work by Levy et al. which is closely related to ours in terms of its goal, but not the approach taken. In [LSK95], the notion of the *world-view* is introduced as a global, fixed domain model of a certain part of the world on which both information providers and consumers must define views. This work is in some sense an approach inverse to ours [RLN97] but relies on the existence of fixed, invariant "world model" which is unlikely to be available in practice. Furthermore, changes to the world model are not possible in this approach or would require a manual redefinition of both information providers' and consumers' queries.

Related approaches to Levy's are Arens et al. [AKS96] and the SoftBot project [EW94]. The SoftBot project has a different approach to query processing as they assume that the system has to discover the "links" among data sources that are described by action schemas. While related to our view synchronization algorithm CVS [NLR98], the SoftBot planning process also relies on discovering connections among information sources when very different source description languages are used. Neither SIMS nor SoftBot address the particular problem of evolution under capability changes of participating information sources.

3 Background: The EVE-System

Since our work is based on the *EVE*-Project [RLN97], we give a short overview over those elements of the EVE-System that we need as background in this paper. The algorithm described in this paper builds on top of the *one-step* synchronization algorithms as described in the *EVE*-Project [RLN97, NLR98, KRH98]. As we will see, the new algorithm in some cases falls back on these old algorithms and performs one or a sequence of single-step view synchronizations.

E-SQL or *Evolvable-SQL* is an extension of SQL that allows the view definer to express preferences for view evolution [LNR97]. A user defining a view can specify what information is dispensable and what information is replaceable by similar information from other information sources (ISs), and whether a changing view extent is acceptable. Those user preferences are expressed as parameters that can be attached to basic SQL query elements. For example, the qualifier AD declares an attribute dispensable, RR means a replaceable relation, and $\mathcal{VE}$ (which is one of $\subseteq, \supseteq, \equiv, \approx$) expresses the relationship between the original and rewritten query as required by the view definer. The default for $\mathcal{VE}$ is $\equiv$, for all other parameters it is *false*.

Example 1. A typical E-SQL query would be:

CREATE VIEW Asia-Customer ($\mathcal{VE}$ = "$\supseteq$") *AS*
SELECT *Name, Address, Phone* ($\mathcal{AD} = true$, $\mathcal{AR} = true$)
FROM *Customer C* ($\mathcal{RR} = true$), *FlightRes F*
WHERE (*C.Name = F.PName*) *AND* (*F.Dest = 'Asia'*) ($\mathcal{CD} = true$)

View definitions are stored in a *View Knowledge Base (VKB)*. Once a view is defined, *EVE* view synchronization algorithms are invoked after schema changes in all ISs participating in this view and attempt to find replacements for missing view elements. For the purpose of identifying view element replacements from other ISs, we express relationships between ISs through constraints (e.g., agreeing data types, functional dependencies between attributes, and extent overlaps between relations, all of which are stored in the *Meta Knowledge Base (MKB)*). One important constraint is the *containment-constraint* between two relations, stating that a (horizontal and/or vertical) fragment of a relation is semantically contained or equivalent to a (horizontal and/or vertical) fragment of another. Examples of view synchronization can be found in the technical report to this paper [KR00].

Since view synchronization algorithms may generate many possible query rewritings, one needs to be selected as the new view definition. For this purpose we have developed the QC-Model [LKNR99] to estimate the *quality* and *cost* of the rewritings. Each legal query rewriting will in general preserve a different amount (extent) and different types (interface) of information, which we measure through a metric that we have defined on views and refer to as the *quality* of the view. Also, each new view query will cause different *view maintenance costs*, since in general data will have to be collected from a different set of ISs. With these two dimensions, the QC-Model can assign a numerical *QC-Value* to each query rewriting and thus *compare* different rewritings with each other.

4 Modeling Meta Data and Meta Data Changes

4.1 Meta Data Model

In the *EVE*-System as introduced by [RLN97], View and Meta Knowledge Bases (VKB and MKB) exist to maintain data warehouses over changing information sources. MKB data is encoded in the form of schema definitions and constraints. In this sense, we will describe schemas of underlying information sources by Attribute Constraints and Relation Constraints, and for the purpose of this paper, we will concentrate on a total of four different constraints that comprise our MKB: *attribute constraints* ($\mathcal{AC}$) define attribute names, *relation constraints* ($\mathcal{RC}$) define relation names, *containment constraints* ($\mathcal{CC}$) define relationships between relations, and *containment pair constraints*($\mathcal{CPC}$) relate attributes in those relations to each other (i.e., define projections of relations who are in a set relationship to each other). We also introduce two functions that relate attributes with their relations (rel: $\mathcal{AC} \rightarrow \mathcal{RC}$) and $\mathcal{CPC}$s with their $\mathcal{CC}$s (con: $\mathcal{CPC} \rightarrow \mathcal{CC}$). For details, refer to our technical report [KR00].

4.2 Changes in Meta Data

Changes to the schemas of sources which are modeled in the Meta Knowledge Base cause MKB and VKB changes. This paper describes how to synchronize view definitions with such *meta data changes* under consideration of previous meta data changes in the information space, and how to make use of added meta knowledge by "re-installing" a previously deleted schema element. In order to better specify the meaning of "previously deleted element", we introduce the concept of *history* of view and meta knowledge to our system, starting with a formal definition of schema changes.

Definition 1 (meta data change). *Let $\mathcal{C}_i$ be a constraint ($\mathcal{AC}$, $\mathcal{RC}$, $\mathcal{CC}$, or $\mathcal{CPC}$, cf. Section 4.1). A* meta data change *$\mathcal{MDC}_i$ is the addition or deletion of $\mathcal{C}_i$ to or from the Meta Knowledge Base, transforming the MKB from a previous state MKB_i to a subsequent state MKB_{i+1}. Valid meta data changes are:* delete-$\mathcal{CPC}$ ($\mathcal{C}_i$), delete-$\mathcal{CC}$ ($\mathcal{C}_i$), delete-$\mathcal{RC}$ ($\mathcal{C}_i$), delete-$\mathcal{AC}$ ($\mathcal{C}_i$), add-$\mathcal{CPC}$ ($\mathcal{C}_i$), add-$\mathcal{CC}$ ($\mathcal{C}_i$), add-$\mathcal{RC}$ ($\mathcal{C}_i$), add-$\mathcal{AC}$ ($\mathcal{C}_i$). *Each $\mathcal{MDC}_i$ has as a parameter exactly one constraint in the MKB which is denoted by* constraint($\mathcal{MDC}_i$).

We also introduce the notion of an *inverse* meta data change operation. Two meta data changes are *inverse* to each other if they are add- and delete-operations, respectively, and refer to the same constraint (e.g. *delete-*$\mathcal{RC}(\mathcal{RC}_1) = (\textit{add-}\mathcal{RC}(\mathcal{RC}_1))^{-1}$).

4.3 Evolution of the MKB under Meta Data Changes

For the eight operations in Definition 1, the respective constraint is simply added to or removed from the MKB. In addition to that, some changes imply others. For example, the meta data change operations *delete-*$\mathcal{RC}$ and *delete-*$\mathcal{CC}$ trigger deletions of Attribute Constraints and Containment Pair Constraints, respectively. Those implicit operations are executed on the MKB immediately after the meta data change is detected, but before the view synchronization algorithm is run on the view definitions in the VKB. Similarly, we assume (or generate) subsequent additions of sub-constraints in the same fashion for *add-*$\mathcal{RC}$ and *add-*$\mathcal{CC}$-operations. In the case that the set of attributes in a relation has changed while this relation was unavailable, we assume that appropriate *add-*$\mathcal{AC}$ or *delete-*$\mathcal{AC}$ operations are generated after the *add-*$\mathcal{RC}$ operation. Note that renames of relations or attributes cause all $\mathcal{RC}$ or $\mathcal{AC}$ in all MKB states to be renamed at once and do not constitute a $\mathcal{MDC}$ in the sense of Definition 1.

4.4 Dependency of Views on Meta Data

In previous view synchronization algorithms [RLN97, NLR98, KRH98] the rewriting of a view is accomplished with the help of constraints that state (partial) redundancy in the information space. For instance, a containment constraint between R and S can be used to replace relation R by relation S in a view if

the former becomes unavailable. The meaning of that constraint is that the information provider(s) of relations R and S guarantee that the constraint is and remains valid. Even when relation R becomes unavailable, this condition will still be kept valid. If this $\mathcal{CC}$-constraint would also be deleted from the MKB, it could not be utilized by the view synchronization process. We define a rewritten view $V^{(1)}$ to be *constraint-dependent* on a $\mathcal{CC}$-constraint if the latter was used in the above sense in the rewriting of a view V. A formal definition can be found in [KR00]. Note that any further rewriting of view $V^{(1)}$ can, but does not have to, retain the view's constraint-dependency on any constraint.

4.5 Modeling the History of Meta Data and Views

We will now define a way to model the *history* of meta data and views, i.e., the sequence of changes in meta data that have occurred on a MKB and VKB.

Definition 2 (history graph). *Let a view V be rewritten into $V^{(n)}$ by a meta data change sequence $\mathcal{S} = \{\mathcal{MDC}_1, \mathcal{MDC}_2, \ldots, \mathcal{MDC}_n\}$. The* history $\mathcal{H}(V^{(n)})$ *of view rewriting $V^{(n)}$ is a graph $\mathcal{H} = (N, E)$ with N the set of nodes representing the states of both MKB and VKB at a certain point in time, and E the (labeled directed) edges representing transitions from one state to the next (meta data changes). Each node N_i is a pair $< MKB_i, VKB_i >$ and each edge E_i is labeled with the meta data change $\mathcal{MDC}_i$ it represents. Node N_0 contains the original Meta Knowledge and View Knowledge Bases (i.e., the meta knowledge at view definition time and the originally defined views) while N_n represents the current state of the system.*

5 The History-Driven View Synchronization Algorithm

View synchronization has to be performed when a view is affected after a schema change in the information space. Views whose definitions are still valid after the change do not have to be synchronized. A view V is affected by a schema change if (1) a constraint on which V was constraint-dependent is added to the MKB or (2) a constraint on which V is currently constraint-dependent is removed from the MKB.

The History-Driven View Synchronization Algorithm (HD-VS), which is executed after a schema change for each affected view, uses three main concepts: backtracking in the history of a view, re-applying a part of a meta data update sequence from that history, and reconstructing part of the view's history graph in the process of re-application of meta data changes. Figure 1 gives an example of the algorithm. Four meta data changes are executed on a view V. After each meta data change, HD-VS checks whether the history of the view contains information that could be used to rewrite view V in an appropriate way. Two consecutive *delete-constraint* operations both lead to backtracking in the history of the view but do not find an appropriate inverse operation to cancel with. In both cases, HD-VS falls back on a single-step view-synchronization algorithm

(e.g., POC). The third operation, an *add-constraint* operation, also leads to unsuccessful backtracking but in the case of an addition to the information space, no POC-view synchronization is *necessary* for the view to remain valid. The fourth operation, an *add-operation* inverse to a previous *delete*-operation, leads to successful backtracking. Here, the two operations *add-X* and *del-X* cancel out. HD-VS applies a meta data -sequence to V that does not contain these two operations, but all others. Then, the remaining two operations (*del-Y* and *add-U* in our example) are applied in the above mentioned fashion, but do not lead to further cancellations, so the final view obtained is $V_2^{(2)}$.

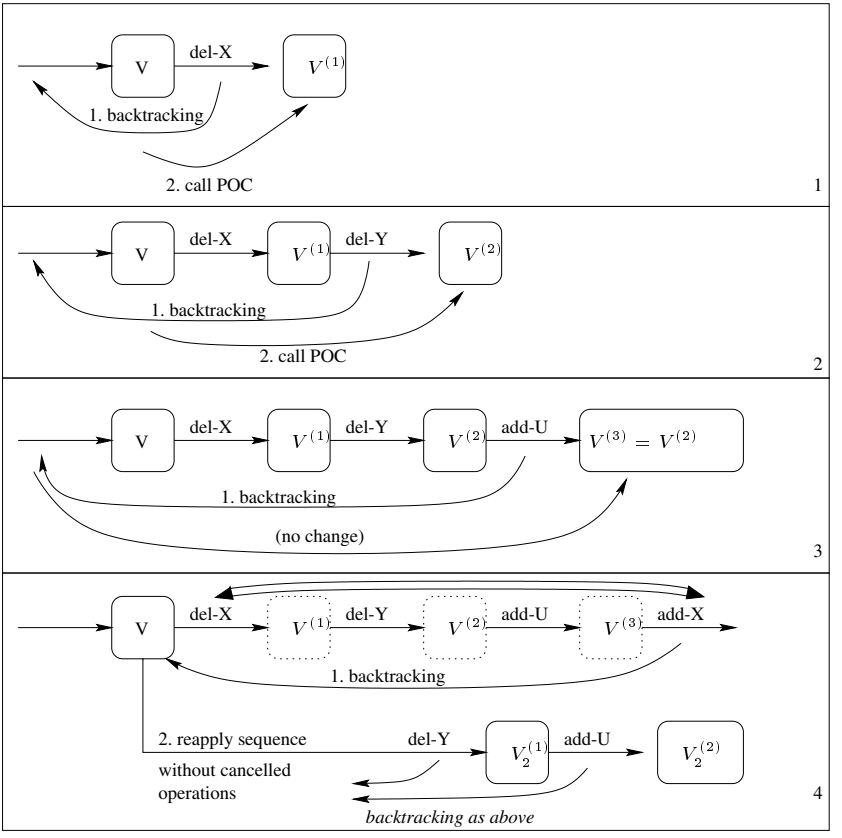


Fig. 1. Example Sequence for the History-Driven View Synchronization Algorithm (HD-VS).

5.1 Backtracking in the History of a View

In order to perform view synchronization, HD-VS applies a technique that we will call *backtracking*. The purpose of the backtracking process is to find a view

V_i in the history $\mathcal{H}(V^{(n)})$ of V from which to re-apply or *redo* a changed meta data update sequence. The idea of backtracking now consists in finding a node in $\mathcal{H}(V)$ for which a certain condition C holds.

Definition 3 (backtracking). *Backtracking in the history $\mathcal{H}(V^{(n)})$ of a view rewriting $V^{(n)}$ after a meta data change $\mathcal{MDC}_{n+1}$ to a constraint $\mathcal{C}_i$ is the traversal of the graph $\mathcal{H}(V^{(n)})$, starting from node N_n through nodes $N_{n-1}, N_{n-2}, \ldots$, until the following condition is satisfied:*

1. *if $\mathcal{MDC}_{n+1}$ is an* add-constr.*-oper.: $V^{(i)}$ is constraint-dependent on $\mathcal{C}_i$ or*
2. *if $\mathcal{MDC}_{n+1}$ is a* delete-constr.*-oper.: $V^{(i)}$ is **not** constraint-dependent on $\mathcal{C}_i$.*

The backtracking process returns the node N_i with the highest index i (i.e., the most recent view) satisfying the above condition.

The motivation behind this definition of backtracking is as follows: If a constraint $\mathcal{C}_i$ is added to the MKB, we want to determine whether this constraint has previously been used in the history of view V. In this case, $\mathcal{C}_i$ must have been deleted from the MKB earlier and we can *cancel* the add with the delete of $\mathcal{C}_i$. Therefore, we backtrack to a view $V^{(i)}$ that used $\mathcal{C}_i$, since $V^{(i)}$ must precede the deletion of constraint $\mathcal{C}_i$. In the opposite case of a delete-constraint-operation on constraint $\mathcal{C}_i$, we look for the last view $V^{(i)}$ that was not constraint-dependent on $\mathcal{C}_i$ since we can then re-apply the remaining schema change sequence from $V^{(i)}$ on under the assumption that $\mathcal{C}_i$ was never part of the MKB.

5.2 Re-applying a Meta Data Update Sequence

After backtracking to a certain view $V^{(i)}$, a meta data change sequence S' derived from the original sequence S has to be applied to $V^{(i)}$. In the first case of *add-constraint*-operations, we cancel out the corresponding previous delete-operation from S and apply this shorter sequence. In the second case of *delete-constraint*-operations, the constraint C is removed from the MKB. The sequence S is then applied to this changed MKB, without C being available for any view synchronization step. In the case of a *delete-$\mathcal{CC}$* or *delete-$\mathcal{RC}$*, all dependent $\mathcal{CPC}$ or $\mathcal{AC}$ are also deleted from the MKB and subsequent add-$\mathcal{CPC}$- or add-$\mathcal{AC}$-operations are removed from S. A formal definition of this process is given in [KR00].

5.3 Reconstructing the History Graph

As the sequence S' is applied to MKB_i, new nodes $N_{i+1}, \ldots, N_j (j \leq n)$ are created and old nodes are discarded from the graph. Lastly, a new node N_{j+1} is added to $\mathcal{H}(V)$, containing MKB_{j+1} and VKB_{j+1} and connected with N_j through an edge E_{j+1} labeled with meta data change $\mathcal{MDC}_{j+1}$.

In the technical report [KR00], we handle the case when backtracking reaches the beginning of the meta-data change sequence. We also analyze details of the different meta data change operations, such as co-dependencies between different operations, and give examples for a number of cases. This is omitted here due to the limited space.

6 Evaluation

In the technical report accompanying this paper [KR00], we give several theorems on the correctness and advantages of the HD-VS algorithm. We prove that (1) after a valid meta data change that does not lead to backtracking, our algorithm will give a valid view, and (2) backtracking and re-applying a meta-data change sequence as discussed in Section 5 will always lead to valid view rewritings. We also show that under both add- and delete-schema-changes, our algorithm will give equal or better view rewritings than previous algorithms.

7 Conclusion

In this paper, we present a solution for view synchronization under meta data updates. With our algorithm, a view that is defined over a given set of relations and attributes can be synchronized with *additions and deletions* of both. Previous approaches [RLN97, NLR98] to this problem could only adapt views to *deletions* in the information, but not to additions, so that, for example, temporary unavailability of information sources could not be accounted for. This technology can be applied to views defined over distributed information sources, especially as those sources are independent and prone to change their schemas or query capabilities.

Acknowledgments. We would like to thank all members of the DSRG research group at WPI. In particular, we are grateful to Xin Zhang for helping to implement the EVE system. We also thank Amy Lee and Anisoara Nica from the University of Michigan for interactions on the EVE project.

References

[AASY97] D. Agrawal, A. E. Abbadi, A. Singh, and T. Yurek. Efficient View Maintenance at Data Warehouses. In *Proceedings of SIGMOD*, pages 417–427, 1997.

[AKS96] Y. Arens, C. A. Knoblock, and W.-M. Shen. Query Reformulation for Dynamic Information Integration. *Journal of Intelligent Information Systems*, 6 (2/3):99–130, 1996.

[BLT86] J. A. Blakeley, P.-E. Larson, and F. W. Tompa. Efficiently Updating Materialized Views. *Proceedings of SIGMOD*, pages 61–71, 1986.

[CS96] S. Chaudhuri and K. Shim. Optimization of queries with user-defined predicates. In T. M. Vijayaraman et al., editors, *International Conference on Very Large Data Bases*, pages 87–98, Los Altos, CA 94022, USA, 1996. Morgan Kaufmann Publishers.

[EW94] O. Etzioni and D. Weld. A Softbot-based interface to the Internet. *Communications of the ACM*, 37(7):72–76, July 1994.

[Fin82] S. Finkelstein. Common subexpression analysis in database applications. In M. Schkolnick, editor, *Proceedings of SIGMOD*, pages 235–245, Orlando, Florida, 2–4 June 1982. ACM, New York.

[GM95] A. Gupta and I. Mumick. Maintenance of Materialized Views: Problems, Techniques, and Applications. *IEEE Data Engineering Bulletin, Special Issue on Materialized Views and Warehousing*, 18(2):3–19, 1995.

[GMHI+95] H. García-Molina, J. Hammer, K. Ireland, et al. Integrating and accessing heterogeneous information sources in TSIMMIS. In *AAAI Spring Symposium on Information Gathering*, 1995.

[JK84] M. Jarke and J. Koch. Query Optimization in Database Systems. *ACM Computing Surveys*, pages 111–152, 1984.

[KR00] A. Koeller and E. A. Rundensteiner. A History-Driven Approach at Evolving Views Under Meta Data Changes. Technical Report WPI-CS-TR-00-01, Worcester Polytechnic Institute, Dept. of Computer Science, 2000.

[KRH98] A. Koeller, E. A. Rundensteiner, and N. Hachem. Integrating the Rewriting and Ranking Phases of View Synchronization. In *Proceedings of the ACM First International Workshop on Data Warehousing and OLAP (DOLAP'98)*, November 1998.

[LKNR99] A. J. Lee, A. Koeller, A. Nica, and E. A. Rundensteiner. Non-Equivalent Query Rewritings. In *International Database Conference*, Hong Kong, July 1999.

[LMSS93] A. Levy, I. S. Mumick, Y. Sagiv, and O. Shmueli. Equivalence, query reachability and satisfiability in datalog extensions. In *Proceedings of the Twelfth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 109–122, Washington, DC, 25–28May 1993.

[LMSS95] J. J. Lu, G. Moerkotte, J. Schue, and V. S. Subrahmanian. Efficient maintenance of materialized mediated views. In M. J. Carey and D. A. Schneider, editors, *Proceedings of SIGMOD*, pages 340–351, San Jose, California, 22–25 May 1995.

[LNR97] A. J. Lee, A. Nica, and E. A. Rundensteiner. The EVE Framework: View Synchronization in Evolving Environments. Technical Report WPI-CS-TR-97-4, Worcester Polytechnic Institute, Dept. of Computer Science, 1997.

[LSK95] A. Y. Levy, D. Srivastava, and T. Kirk. Data model and query evaluation in global information systems. *Journal of Intelligent Information Systems - Special Issue on Networked Information Discovery and Retrieval*, 5(2):121–143, 1995.

[NLR98] A. Nica, A. J. Lee, and E. A. Rundensteiner. The CVS Algorithm for View Synchronization in Evolvable Large-Scale Information Systems. In *Proceedings of International Conference on Extending Database Technology (EDBT'98)*, pages 359–373, Valencia, Spain, March 1998.

[NR98] A. Nica and E. A. Rundensteiner. Using Containment Information for View Evolution in Dynamic Distributed Environments. In *Proceedings of International Workshop on Data Warehouse Design and OLAP Technology (DWDOT'98)*, Vienna, Austria, August 1998.

[RKZ00] E. A. Rundensteiner, A. Koeller, and X. Zhang. Maintaining data warehouses over changing information sources. *Communications of the ACM*, June 2000.

[RLN97] E. A. Rundensteiner, A. J. Lee, and A. Nica. On Preserving Views in Evolving Environments. In *Proceedings of 4th Int. Workshop on Knowledge Representation Meets Databases (KRDB'97): Intelligent Access to Heterogeneous Information*, pages 13.1–13.11, Athens, Greece, August 1997.

A Logical Model for Data Warehouse Design and Evolution

Mokrane Bouzeghoub, Zoubida Kedad

Laboratoire PRiSM, Université de Versailles
45, avenue des Etats-Unis 78035 Versailles Cedex, France
email: {Mokrane.Bouzeghoub, Zoubida.Kedad}@prism.uvsq.fr

Abstract. A data warehouse is a software infrastructure which supports OLAP applications by providing a collection of tools for data extraction and cleaning, data integration and aggregation, and data organization into multidimensional structures. At the design level, a data warehouse is defined as a hierarchy of view expressions whose ultimate nodes are queries on data sources. In this paper, we propose a logical model for a data warehouse representation which consists of a hierarchy of views, namely the base views, the intermediate views and the users views. This schema can be used for different design purposes, as the evolution of a data warehouse which is also the focus of this paper.

1 Introduction

Data warehousing is a software infrastructure which supports OLAP applications by providing a collection of tools to (i) collect data from a set of distributed heterogeneous sources, (ii) clean and integrate this data into a uniform representation, (iii) aggregate and organize this data into multidimensional structures and (iv) refresh it periodically to maintain its currency and accuracy. Many problems related with this approach have been addressed, such as data extraction and cleaning, selection of the best views to materialize, updates propagation, and multidimensional representation and manipulation of data.

Several projects dealing with data warehouses such as Whips [6], H2O [23], DWQ [8] have been undertaken. Among the research problem related to data warehousing [20], a substantial effort have been devoted to the selection of the set of views to materialize [1] [17] [18] [14] [22], and the maintenance of the materialized views [21]. Most of the addressed problems are related with the physical design of data warehouses. A very few effort is devoted to the conceptual and logical modeling of a data warehouse. The only abstract representation given to the database which implements a data warehouse is the so-called star-schema or its constellation or snow-flake variants. In our knowledge, the properties of these schemas were not formally stated, and the semantics of these schemas is even not well-defined.

Within the DWQ project [9], a substantial effort has been done toward the definition of a conceptual schema [2]. A description logic approach is used to formally define a

Y. Kambayashi, M. Mohania, and A M. Tjoa (Eds.): DaWaK 2000, LNCS 1874, pp. 178–188, 2000.
© Springer-Verlag Berlin Heidelberg 2000

DW conceptual schema as an extended entity-relationship diagram over which some desired properties are defined. The approach provides a uniform representation and a set of reasoning techniques which allow to correctly specify and validate DW schemas. It also provides a theoretical background to study the complexity of some design problems.

The approach presented in this paper follows the same line although positioned at the operational level instead of at the definition level. We concentrate on the study of a set of views which model the DW as a set of relational queries. We give a characterization of a these views in terms of redundancy. This property can be used as a design rule to enforce the quality of the logical schema. We then show how this model is used to support data warehouse evolution. Some approaches based on a repository [15] give a map-road which allows to trace DW changes. [19] studies the evolution at the materialized views level. This latter approach is based on a specific notion of redundancy which states that a materialized view is redundant if its removal does not affect the rewriting of the users queries, and if its removal does not increase the operational cost of the DW. If a materialized view is redundant, it can be removed from the set of materialized views.

The paper is organized as follows : In section 2, we describe our motivations. Section 3 presents the design process of a data warehouse logical schema. Section 4 presents one possible use of the logical schema: the control of the data warehouse evolution, and a conclusion is given in section 5.

2. Motivations

In database design, the conceptual schema is considered as an abstract representation which captures user needs in terms of data and constraints on this data. Different formalisms have been proposed, ranging from the entity-relationship model with its various extensions to more semantic models which include objects behavior. The logical schema is more operational, without necessarily being a physical implementation. It allows to enforce some logical properties related integrity constraints and updates. The physical schema is considered as the actual structure of the database on which optimization and tuning can be effectively done. To enforce equivalence between the three schemas, mappings are defined between the conceptual and the logical schemas and between the logical and the physical schemas.

A DW is nothing but a database which is less or more complex in its definition. Consequently, applying the previous abstraction levels appears as a natural approach. The main difference with a database used in a classical application is the feeding of this database. The DW is automatically populated from external sources instead of being inserted and updated by user applications. Users are limited to query the DW and to apply analysis tools such as datamining tools. So, with respect to DW systems, the structure mappings existing in a traditional database design process is complemented by operational mappings which consider each abstraction level as a set

of views of the lower level ; each view being defined as a query on lower level schema [2].

Users' needs are represented by a set of views which may be redundant, or may share common sub-expressions. Our goal is to represent this set of views by a single representation at the logical level. We suppose that each view is represented by a relational query, and we propose a logical model for the views of a data warehouse called the view synthesis graph (VSG). The high level views are computed using a set of intermediate views or base views. The base views are directly computed from the relations of a single source. Each intermediate view represents either an expression on which a single relational operation is applied to produce a user view, or an expression shared by two or more intermediate views.

The logical model can be used for many purposes [11] [10] : (i) as an intermediate representation between the conceptual design and the physical design, providing an operational view of the DW without necessarily dealing with performance nor physical representation of data, (ii) as a reference schema from which the physical design starts and to which the benefit of the selected materialized views is balanced, (iii) as a support to control the DW evolution both at its client and source levels.

The logical model provides an independency between user queries and materialized views. Consequently, it allows source and view evolution. We focus here on the control of changes affecting the user needs, data sources or the design choices. For each case, we show how the logical model can be used to determine the effects of these changes.

3. The design approach of the data warehouse logical model

The design process of the DW logical schema starts from the representation of the different user views. We assume that these views are represented by a relational expression. The result of this process is the logical schema, called the view synthesis graph (VSG). One important feature of the VSG is that it shows a set of sub-expressions which are shared by several user views. The design steps of the logical schema are the following :

- The generation of the multi-query graphs (MQGs) : we suppose that each user view V_i is a relational query to which a set of equivalent plans are associated.
- The generation of the multi-view graph (MVG) : the set of user view expressions are integrated into a single graph called the multi-view graph. This graph represents all the MQGs produced during the previous step.
- The generation of the view synthesis graph (VSG) : this graph is generated from the MVG by eliminating redundant expressions. The resulting graph is the logical schema of the data warehouse.

3.1. The multi-query graphs

Let us consider a user view V_i. This view can be represented by a set of equivalent algebraic expressions denoted E_{Vi} and such that: $E_{Vi} = \{e_{1Vi}, e_{2Vi}, ..., e_{kVi}\}$. The algebraic expressions corresponding to the user view V_i are denoted e_{kVi}. They use a set of operations such as join, project and restrict operations, or some aggregate function. Each expression represents a given plan for the user view V_i. In order to limit the number of considered plans, we will consider only those for which the project operation is the last one.

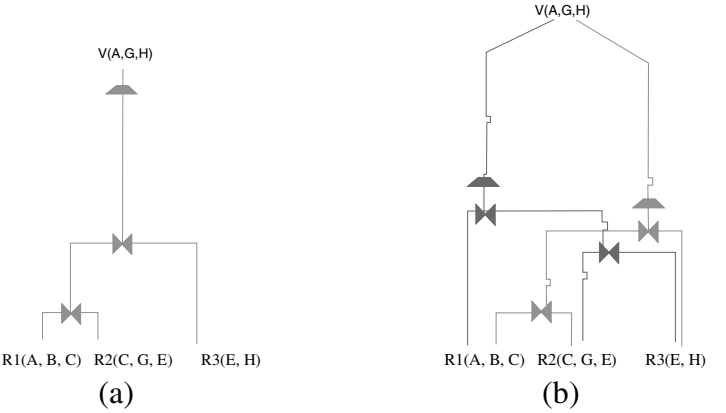


Fig. 1. The multi-query graph associated with a user view

Given a user view V_i and the associated set of equivalent expressions E_{Vi}, the MQG represents each element e_{jVi} of the set E_{Vi}. The MQG does not represent all the equivalent expressions to the initial view, because the search of these expressions may lead to an infinite set of expressions. For example, in the query given figure 1a, computed from the source relations R_1, R_2 and R_3, the multi-query graph (figure 1b) is composed of two expressions representing the two possible orderings of the join operations.

3.2. The multi-view graph

Each user view being represented by a MQG, the multi-view graph (MVG) consists in integrating all the MQGs into a single graph. More formally, if $V = \{V_1, V_2, ..., V_n\}$ represents the set of user views, and $E_{Vi} = \{e_{1Vi}, e_{2Vi}, ..., e_{kVi}\}$ the set of equivalent expressions corresponding to a view V_i. The MVG is such that: $MVG = \{E_{V1}, E_{V2}, ..., E_{Vn}\}$

Figure 2 shows an example of the graphical representation of an MVG, built by integrating the MQGs corresponding to three user views V_1(A, B, C, D, E, F), V_2(A, G, H) and V_3(A, B, E, F). In this example, we can notice that some sub-expressions are used by more than one user view. An ellipse surrounds these expressions.

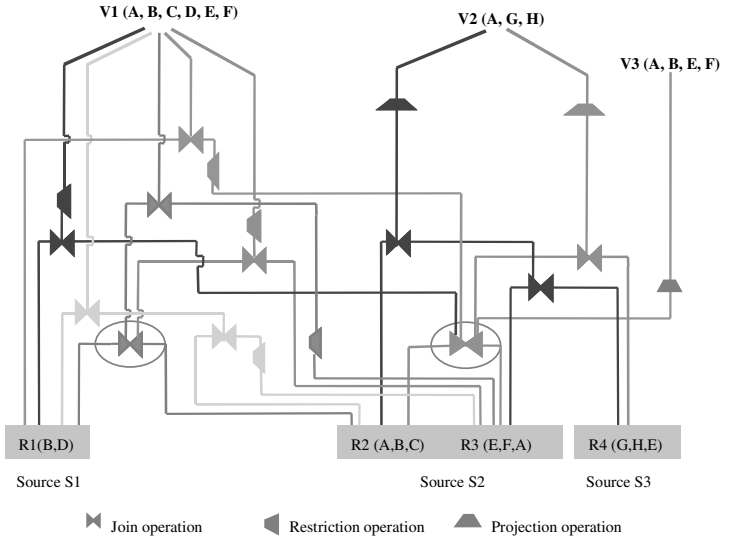


Fig. 2. Example of a multi-view graph (MVG)

The MVG graph represents a set of equivalent expressions for all the user views of the DW. If we consider the graphical representation of a MVG, each node in the graph represents either a source relation, a sub-expression, or an expression corresponding to a user view. The graph shows all the existing sub-expressions used by each user view. Each sub-expression may be either shared by several plans, or used in only one plan. The sub-expressions used by more than one user view are surrounded by an ellipse.

3.3. The view synthesis graph

The view synthesis graph (VSG) is a canonical representation derived from the MVG. This representation is such that each user view is represented by only one plan. For each user view, among all the plans of the MVG, only one will be kept in the VSG. The plans of the VSG are chosen according to the two following criteria : (i) the maximal number, denoted p, of user views which share the same sub-expression in the MVG, (ii) and the number of shared sub-expressions, denoted n.

In the example given in figure 2, the shared sub-expressions are surrounded by an ellipse. The first of these sub-expressions, the join operation between R_1 and R_2, is shared by two different plans of the same user view V_1; the second of these sub-expressions, the join operation between R_2 end R_3, is shared by plans representing the three views V_1, V_2, and V_3. The maximal number of user views which share the same sub-expression is then 3. Intuitively, building a VSG by maximizing p will result in a graph which maximizes the number of user views which share a sub-expression, while building a VSG by maximizing n will result in a graph which maximizes the number of shared sub-expressions.

In the remaining of this section, we will present some definitions to characterize a VSG in terms of a level of overlapping.

3.3.1. Level of dependency of a given expression

The level of dependency of an expression e_{jVi} representing a user view V_i in a multi-view graph MVG is the number of distinct user views V_l such that there exists an expression e_{kVl} which share a sub-expression *e* with e_{jVi}. This notion of dependency allows to define the redundant plans in a MVG.

3.3.2. Redundant plans

A plan e_{Vi} is redundant in the MVG if there exists a distinct plan e'_{Vi} representing the same view V_i such that *level_of dependency*(e_{Vi}) < *level_of dependency*(e'_{Vi}). If e'_{Vi} is the plan having the highest level of dependency $n(e'_{Vi})$, all plans having a lower level of dependency are said to be redundant. Given this notion of redundancy, a view synthesis graph (VSG) is defined as a graph representing only one expression for each user view, and in which each plan e_{Vi} representing the user view V_i is not redundant. By definition and construction, the multi-view graph is unique. The view synthesis graph is not unique. For a given view V_i, there may exist more than one plan having the highest level of dependency, and consequently, the same MVG may lead to more than one VSG.

3.3.3. Level of overlapping of a view synthesis graph

The level of overlapping of a VSG is the maximal level of dependency of the expressions represented in this graph. Consider that the VSG contains a number k of users views V_i described by their plans denoted e_{Vi}, the level of overlapping of this VSG, denoted *n*(VSG) is such that $n(VSG) = Max(n(e_{Vi}), i= 1, k)$. The VSG is a hierarchy of views. Besides the user views, this hierarchy contains intermediate views and base views defined as follows :

- Each sub-expression computed from the relations provided by a single data source is defined as a base view.
- A given sub-expression in the VSG is an intermediate view if it satisfies the following conditions :
    1. this sub-expression is shared by more than one other expression in the VSG
    2. this sub-expression is neither a base view, nor a user view.

Starting from the MVG given in figure 2, the VSG is built by (i) eliminating the plans which do not share any sub-expression with other user views, and (ii) by eliminating redundant plans, that is plans which level of dependency is lower than the one of a distinct plan representing the same user view. The resulting VSG is given in figure 3. Each non-redundant plan corresponding to a user view V is represented by an algebraic tree. The expressions RB_1, RB_2 and RB_3 are the base views. Each of them is computed from the data provided by a single data source. These expressions can be viewed at as the extractors associated with each source. The expression RB_2 is an intermediate view shared by the three user views V_1, V_2 and V_3.

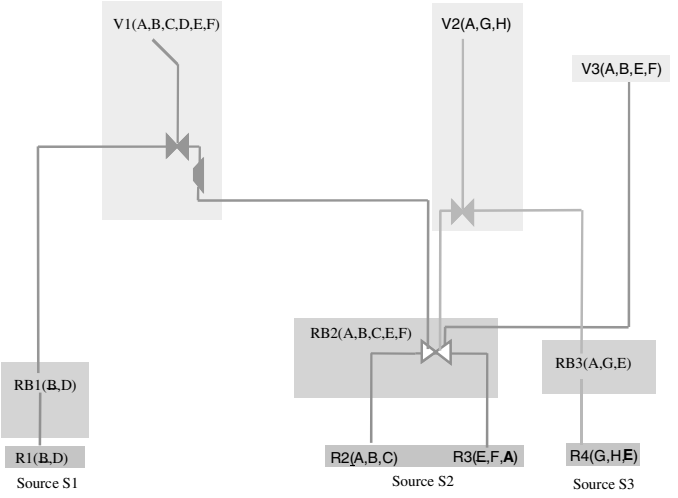


Fig. 3. Example of a logical model

4. Data warehouse evolution

The logical model of a DW represents a canonical representation of the user views. One possible use of this representation is for controlling the DW evolution. We consider hereafter three distinct cases of possible evolutions: the evolutions of the user views, the evolution of the data sources, and the evolution of the materialized views. In each of these situations, the view synthesis graph can be used in order to evaluate the impact of such changes in the DW environment.

4.1. Evolution of the users needs

Two situations may occur with respect to user needs: a user view can be either added or removed from the data warehouse.

Adding a new view : If a new view V_i is added to the data warehouse, the VSG can be used to check if this view can be derived or not from the hierarchy of views represented by the logical model. Three situations may occur :

- V_i can be derived directly from the VSG if it satisfies the following criteria : (i) the expression representing V_i is either a base view or an intermediate view of the VSG, (ii) the expression representing V_i is a sub-expression of the VSG,
- V_i can be derivable indirectly from the VSG if it is derivable from one of its transformations, that is if it is derivable from the corresponding MVG. Notice that the MVG is always derivable from the VSG.

- V_i is not derivable from the VSG, in this case V_i is added to the VSG ; the expression representing V_i which is added to the VSG is a non-redundant plan. Each sub-expression of V_i shared by other views is a new intermediate view, and each sub-expression computed from the data provided by a single source is a new base view.

Deleting an existing view : If a view V_i is deleted from the data warehouse, the corresponding expression is removed from the view synthesis graph. This operation may lead to the change of some intermediate views or base views because some of the initial intermediate views may become not shared, and some of the initial base views may become useless, if they were used only for computing the view V_i.

4.2. Evolution of data sources

Another change that may occur in the data warehouse environment is adding or deleting a data source. A new source can be added to the environment, and an existing source may be disabled. Each of these situations has some effects on the DW environment; these effects can be determined using the view synthesis graph.

Adding a new data source: If a new data source is added to the data warehouse environment, two situations may occur :

- the new source S_j has the same object types as an existing source S_i; in this case, the VSG may be changed. Different modifications can be done depending on the way users want to integrate this source in their existing views. Union, intersection and difference are candidate operations to integrate the new source. This leads to the change of some intermediate and base views. For example, consider the logical model given in figure 3 and suppose a new source S which contains a source relation R(B, D) is added. R and R_1 have the same schema and the semantics of the application needs the merge of these two relations. The changes occurring to the VSG are shown in figure 4.

The view V_1 which was computed from the relation R_1 of source S_1 is now computed from the union of the two source relations R_1 and R. The union operator is not a shared operator, and the set of intermediate views remains identical. But a new base view RB is introduced in the VSG to represent the extraction of the new source relation R.

- the new source S_j is different from any of the existing sources : in this case, the view synthesis graph is not changed since none of object types in S_j is used by the existing views. The new source S_j can be used only by the newly added views (updates of existing views are considered as definition of new views).
  Deleting a data source

If an existing data source is deleted from the DW environment, it may become impossible to compute some views. These views are identified using the VSG. Consider for example that the source S_3 is deleted in the DW described by the logical model given in figure 4. It becomes impossible to compute the view V_2, which uses

the source relation R_3. If the source S is deleted, the view V_1 is still computable using the source S_1. Deleting a source can modify the sets of base views or intermediate views. For example, if the source S is deleted, RB is deleted from the set of the base views. In general, the effects of deleting a data source S_i in the DW environment are determined by identifying each view V of the VSG that uses a relation R_j contained in this source. If there is a distinct source relation R_k such as R_k and R_j have the same schema, then view V is still computable; if not, the view V becomes not computable.

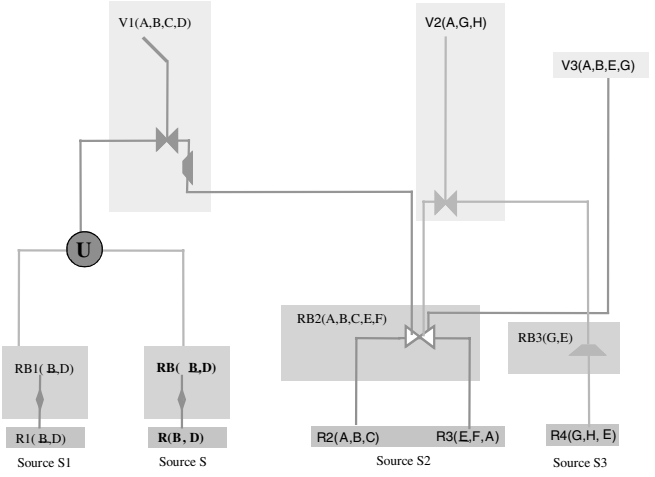


Fig. 4. Adding a source to the data warehouse environment

4.3. Evolution of the materialized views

Another change which can affect the DW is the evolution of the physical schema, i.e. the materialized views. Materialized views are selected on the basis of a cost function which models a compromise between the cost of maintaining the materialized views and the cost of evaluating the virtual views. This cost takes into account a set of parameters such as access frequency, storage space and update propagation cost. When one of these parameters changes, the actual materialized views are not satisfactory with respect to performance objectives. Then a new physical design is restarted and a new set of materialized views is selected. Changing the materialization may result in changing all the expressions defined over the first set of materialized views, unless these expressions were defined on the view synthesis graph which is then used as a logical schema. The existence of the VSG provides a logical independence between user applications and the data warehouse physical schema. The changes on the physical schema leads to changes in the expressions which define the mappings between the logical views (the VSG) and the materialized views.

5. Conclusion

In this paper we have presented a logical reference schema and its use for controlling the evolutions of the DW environment. This reference schema is a canonical representation of data warehouses views. This logical model is built from subsets of the possible plans for each user view, which are integrated in an intermediate representation called the multi-view graph. The VSG is built by identifying for each view one plan which is not redundant in the multi-view graph. The VSG is useful for different purposes, among them the control of the evolution of the data warehouse. We have presented three different kinds of changes that may occur in a data warehouse environment, namely changes affecting the users needs, changes affecting the data sources and changes affecting the physical design. In each situation, we have shown how the view synthesis graph can be used to determine the effects of each of these evolutions.

References

[1] E. Baralis, S. Paraboschi, E. Teniente. Materialized View Selection in a Multidimensional Environment. In *Proceedings of the 23rd International Conference on Very Large Data Bases (VLDB'97)*, Athens, Greece, August 1997.

[2] D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi, R. Rosati. Source Integration integration : conceptual modeling and reasoning support. In *Proceedings of the 3rd IFCIS International Conference on Cooperative Information Systems (CoopIS'98)*, New York City, USA, August 1998.

[3] S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, J. Widom. The TSIMMIS project : Integration of Hererogeneous Information Sources. In *Proceedings of the 10th Meeting of the Information Processing Society of Japan (IPSJ'94)*, Tokyo, Japan, October 1994.

[4] S. Chaudhuri, A. Dayal. An Overview of Data Warehousing and OLAP Technology. Tutorial, In *Proceedings of the 22th International Conference on Very Large Data Bases (VLDB'96)*, Mumbai, India, September 1996.

[5] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajamaran, Y. Sagiv, J. Ullman, V. Vassalos, J. Widom. The TSIMMIS approach to mediation: Data model and Languages. In *2nd International Workshop on Next Generation Information Technologies and Systems (NGITS '95)*, Naharia, Israel, June 1995.

[6] J. Hammer, H. Garcia-Molina, J. Widom, W. Labio, Y. Zhuge. The Stanford Data Warehousing Project. *IEEE Data Engineering Bulletin, Special Issue on Materialized Views and Data Warehousing*, 18(2), June 1995

[7] R. Hull, G. Zhou. A Framework for Supporting Data Integration Using the Materialized and Virtual Approches. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Montreal, Canada, June 1996.

[8] M. Jarke, Y. Vassiliou. Foundation of Data Warehouse Quality : an Overview of the DWQ project . In *Proceedings of the 2nd International Conference on Information Quality*, Cambridge, Mass, 1997.

[9] M. Jarke, M. Lenzerini, Y. Vassiliou, P. Vassiliadis. (edits). Fundamentals of Data Warehouses. Springer, 1999.

[10] Z. Kedad. Techniques d'intégration dans les systèmes d'information multi-source. *PhD Thesis*, Laboratoire PriSM, University of Versailles, France, January 1999.

[11] Z. Kedad, M. Bouzeghoub. Conception de systèmes d'information multi-source. In *Proceedings of INFORSID'99*, Toulon, France, June 1999.

[12] A.Y. Levy, A. Rajaraman, J.J. Ordille. Query-Answering Algorithms for Information Agents. In *Proceedings of the 13th National Conference on Artificial Intelligence and 8th Innovative Applications of Artificial Intelligence Conference (AAAI 96, IAAI 96)*, Portland, Oregon, August 1996.

[13] A.Y. Levy, A. Rajaraman, J.J. Ordille. Querying Heterogeneous Information Sources Using Source Description. In *Proceedings of the 22th International Conference on Very Large Data Bases (VLDB'96)*, Mumbai, India, September 1996.

[14] S. Ligoudistianos, T. Sellis, D. Theodoratos, Y. Vassiliou. Heuristic Algorithms For Designing The Data Warehouse with SPJ Views. In *Proceedings of the International Conference on Data Warehousing and Knowledge Discovery (DAWAK'99)*, 1999.

[15] C. Quix. Repository support for Data Warehouse Evolution. In *Proceedings of the International Workshop on Design and Management of Data Warehouses (DMDW'99)*, Heidelberg, Germany, June 1999.

[16] D. Theodoratos, T. Sellis. The Data Warehouse Configuration. In *Proceedings of the 23rd International Conference on Very Large Data Bases (VLDB'97)*, Athens, Greece, August 1997.

[17] D. Theodoratos, T. Sellis. Data Warehouse Schema and Instance Design. In *Proceedings of the 17th International Conference on Conceptual Modeling (ER'98)*, Singapore, November 1998.

[18] D. Theodoratos, S. Ligoudistianos, T. Sellis. Designing the Global Data Warehouse with SPJ Views. In *Proceedings of the 11th Conference on Advanced Information Systems Engineering (CAISE'99)*, Heidelberg, Germany, June 1999.

[19] D. Theodoratos. Detecting Redundancy in Data Warehouse Evolution. In *Proceedings of the 18th International Conference on Conceptual Modeling (ER'99)*, Paris, France , november 1999.

[20] J. Widom. Research Problems in Data Warehousing. In *Proceedings of the 4th International Conference on Information and Knowledge Management (CIKM'95)*, Baltimore, Maryland, November 1995.

[21] J.L. Wiener, H. Gupta, W.J. Labio, Y. Zhuge, H. Garcia-Molina, J. Widom. A System Prototype for Warehouse View Maintenance. In VIEWS 96

[22] J. Yang, K. Karlapalem, S. Li. Algorithms for materialized view design in data warehousing Environment. In *Proceedings of the 23rd International Conference on Very Large Data Bases (VLDB'97)*, Athens, Greece, August 1997.

[23] G. Zhou, R. Hull, R. King, J.C. Franchitti. Data Integration and Warehousing using H2O. *IEEE Data Engineering Bulletin, Special Issue on Materialized Views and Data Warehousing*, 18(2), June 1995

An Alternative Relational OLAP Modeling Approach

Andreas Bauer, Wolfgang Hümmer, Wolfgang Lehner
University of Erlangen-Nuremberg (Database Systems)
Martensstr. 3, Erlangen, 91058, Germany

EMail: {andreas.bauer, huemmer, lehner}@informatik.uni-erlangen.de

Abstract. Schema design is one of the fundamentals in database theory and practice as well. In this paper, we discuss the problem of locally valid dimensional attributes in a classification hierarchy of a typical OLAP scenario. In a first step, we show that the traditional *star and snowflake schema* approach is not feasible in this very natural case of a hierarchy. Therefore, we sketch two alternative modeling approaches resulting in practical solutions and a seamless extension of the traditional *star and snowflake schema* approach: In a pure relational approach, we replace each dimension table of a *star / snowflake schema* by a set of views directly reflecting the classification hierarchy. The second approach takes advantage of the object-relational extensions. Using object-relational techniques in the context for the relational representation of a multidimensional OLAP scenario is a novel approach and promises a clean and smooth schema design.

1 Introduction

In the last few years, *"Online Analytical Processing"* (OLAP, [CoCS93]) has become a major research area in the database community (special data models: [VaSe99]; SQL extensions: [GBLP96], [SQL99]). One consequence of the OLAP fever is the rejuvenation of the multidimensional data model. The *ROLAP* approach ("Relational OLAP") simulates the multidimensionality and performs data access on top of a relational database engine, thus using sophisticated relational base technology to handle, i.e. store and analyze the typical large data volumes of the underlying data warehouses. This approach however needs an adequate relational representation, which is typically a variation of a *star / snowflake schema*.

Based on the experiences from an industrial project, we have seen that the traditional modeling techniques for the relational based solution, *star* and *snowflake schema*, are not always adequate ([ALTK97]). Even considering multiple variations with regard to slowly changing dimensions, factless fact tables, etc. ([Inmo96]), we demonstrate a modeling problem which is not addressed adequately in literature. This paper reviews our schema design problems with the traditional techniques and proposes two more general and therefore alternative modeling approaches.

The key idea of the multidimensional model is that each dimension of a multidimensional data cube, e.g. products, shops, or time, can be seen as part of the primary key, defining the cartesian product with the elements of dimensions. Consequently, any combination of the composite primary key identifies exactly a single cell within the cube.

Y. Kambayashi, M. Mohania, and A M. Tjoa (Eds.): DaWaK 2000, LNCS 1874, pp. 189–198, 2000.
© Springer-Verlag Berlin Heidelberg 2000

Each cell may hold a numerical fact value (measure) or a NULL value if no such entry exists. As illustrated in figure 1, based on the dimensional elements, e.g. single articles in the product dimension, classifications can be defined to identify different classes C like product families, groups, or product areas. Each classification node at a specific classification level can be seen as an instance of a corresponding classification attribute (CA_i).

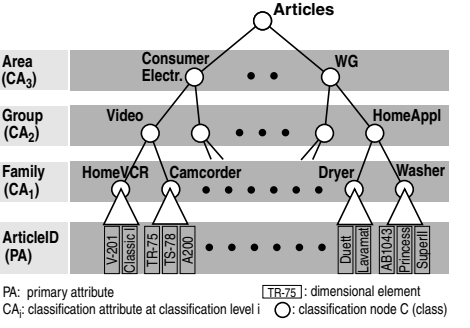


Fig. 1. Sample classification hierarchy

Additionally, dimensional attributes (DA_k) like brand, color, shoptype etc. can be used to enrich the multidimensional analysis process. As depicted in figure 2, these attributes, *characterizing* single dimensional elements, are standing orthogonal with regard to the classification hierarchy, *classifying* dimensional elements.

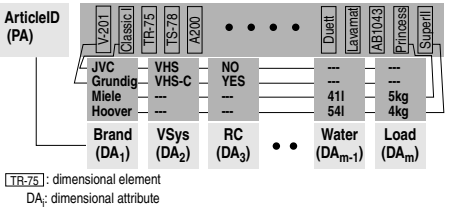


Fig. 2. Sample dimensional attributes

Thus, a typical question according to a multidimensional scenario could be as follows: *Give me the total sales of consumer electronics goods for Europe and the first quarter of 1997 by different brands and different shop types*.

It is worth mentioning here that such a structure with special (better: locally valid) dimensional attributes for different classes within the classification hierarchy reflects the basic idea of classification. What else should be the reason for classification? Classifications are generally used to hide specialities of subclasses and perform an abstraction process when going from subclasses to super-classes. We think that this idea should be adequately reflected in the relational schema design of a multidimensional scenario.

2 The Traditional Relational OLAP Approach

To illustrate the failure of the traditional ROLAP approach and to motivate our alternative approach, we refer to a sample scenario, stemming from a joint research project with a worldwide operating retail research company. In their business, facts like sales, stock, or turnover values of single articles in single shops at a specific period of time are monitored and collected to form the raw database. For example:

```
Facts(ArticleID, ShopID,  Period,  SALES,  STOCK,  TURNOVER)
      TR-75       203      05/97    121     78       333
      TS-78       203      05/97    112     63       121
      .......
```

Generally, this raw database is commonly called a *fact table* and consists of two main components: A set of dimensions, we denote as 'primary attributes' PA_i ($1 \le i \le n$) forming the composite primary key of the table and a set of measures $\{f_1, ..., f_k\}$ denoting the

figures being analyzed. The number of primary attributes n determines the dimensionality of the problem:

$$Facts(\underline{PA_1},...\underline{1},\underline{PA_n}, f_1,..., f_k)$$

In a further step of the retail research evaluation phase, the raw database is analyzed in two ways: On the one hand, the data is aggregated along a predefined classification hierarchy (like the one in figure 1). On the other hand, the data is split into characteristic features of the single articles or shops. For example, each shop holds country specific information about its purchase class or its shop type ("cash&carry", "retail", "hypermarket"). In the product dimension, each article of the 250.000 monitored products belongs to one of 400 product families. Furthermore, each article is characterized by five attributes valid for all products ('*brand'*, '*package type'*, ...) and about 15 attributes which are valid only in the product family or product group to which the article belongs to ('*video system'* only for the product group *Video* equipment, '*water'* usage only for the product family *Washers*).

2.1 Star Schema

The simplest traditional way to model this qualifying information skeleton used during the analysis process is to use a single *dimension table* D^i ($1 \le i \le n$) for each dimension to resolve high-level terms according to the classification hierarchy and to represent dimensional attributes. Since each dimension is connected to the corresponding primary key of the fact table, the whole scenario looks like a 'Star'. Figure 3 illustrates the star schema for the ongoing example. Formally, the schema of the dimension table for the dimension i consists of the primary attribute PA_i, all classification attributes CA_j ($1 \le j \le p$) and the complete set of dimensional attributes DA_k ($1 \le k \le m$).

$$D^i(\underline{PA_i}, DA_1, ...,DA_m, CA_1,...,CA_p)$$

It is worth to note that (mainly for performance reasons) the classification hierarchy is modeled as a set of functionally dependent attributes ($CA_j \rightarrow CA_{j+1}$ ($1 \le j < p$)). Furthermore, the distinction of dimensional elements organized in hierarchies and dimensional attributes further characterizing the elements explicitly prescribed in the multidimensional model gets totally lost. Figure 4 shows the product dimension table for the market research example.

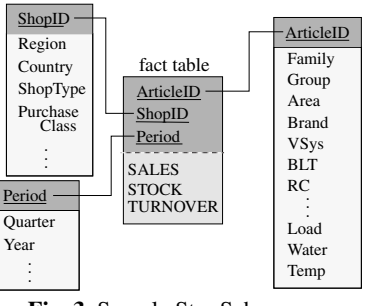


Fig. 3. Sample Star Schema

	PA	DA_1	DA_2			...			DA_m	CA_1	CA_2	CA_3
Articles	ArticleID,	Brand,	VSys,	BLT,	RC,	...	Load,	Water,	Temp,	Family,	Group,	Area)
	TR-75	Sony	HI8	2h	NULL	...	NULL	NULL	NULL	Camcorder	Video	ConsElectr
	TS-78	Sony	HI8	2h	NULL	...	NULL	NULL	NULL	Camcorder	Video	ConsElectr
	A200	JVC	N8	3h	NULL	...	NULL	NULL	NULL	Camcorder	Video	ConsElectr
	V-201	JVC	VHS	NULL	Yes	...	NULL	NULL	NULL	HomeVCR	Video	ConsElectr
	ClassicI	Grundig	VHS-C	NULL	No	...	NULL	NULL	NULL	HomeVCR	Video	ConsElectr
	....											
	AB1043	Ariston	NULL	NULL	NULL	...	5kg	451	NULL	Washer	HomeAppl	WhiteGoods
	Princess	Miele	NULL	NULL	NULL	...	5kg	411	NULL	Washer	HomeAppl	WhiteGoods
	SuperII	Hoover	NULL	NULL	NULL	...	4kg	541	NULL	Washer	HomeAppl	WhiteGoods
	Duett	Miele	NULL	NULL	NULL	...	6kg	NULL	37°C	Dryer	HomeAppl	WhiteGoods
	Lavamat	AEG	NULL	NULL	NULL	...	6kg	NULL	39°C	Dryer	HomeAppl	WhiteGoods
	....											

Fig. 4. Sample star schema dimension table for the product dimension

2.2 Snowflake Schema

Elimination of the functional dependencies in the single dimension tables, i.e. normalizing the *star schema*, leads to small satellite tables representing the dimensional hierarchy. Therefore, the set of dimension tables for a single dimension is modeled in the following manner:

$$D^i(\underline{PA_i}, DA_1, ..., DA_{n_1}, CA_1)$$
$$D^i_1(\underline{CA_1}, DA_{n_1+1}, ..., DA_{n_2}, CA_2)$$
$$\ddots$$
$$D^i_{p-1}(\underline{CA_{p-1}}, DA_{n_{p-1}+1}, ..., DA_m, CA_p)$$

Again, PA_i $(1 \le i \le n)$ denotes the primary attribute for the i-th dimension for joining the fact table. The classification attribute CA_j forming the hierarchy acts as a foreign key in classification level j-1 and as a primary key in level j. Furthermore, all dimensional attributes DA_k at level j are fully dependent on CA_j. To adopt the dimension table for the ongoing example, the classification attributes 'Group' and 'Area' are shifted into two new relations. The relational schema for the product dimension of the current example is depicted in figure 5:

Articles(ArticleID,	Brand,	VSys,	BLT,	RC,	...	Load,	Water,	Temp,	Family)
TR-75	Sony	HI8	2h	NULL	...	NULL	NULL	NULL	Camcorder
TS-78	Sony	HI8	2h	NULL	...	NULL	NULL	NULL	Camcorder
A200	JVC	N8	3h	NULL	...	NULL	NULL	NULL	Camcorder
V-201	JVC	VHS	NULL	Yes	...	NULL	NULL	NULL	HomeVCR
ClassicI	Grundig	VHS-C	NULL	No	...	NULL	NULL	NULL	HomeVCR
....									
AB1043	Ariston	NULL	NULL	NULL	...	5kg	45l	NULL	Washer
Princess	Miele	NULL	NULL	NULL	...	5kg	41l	NULL	Washer
SuperII	Hoover	NULL	NULL	NULL	...	4kg	54l	NULL	Washer
Duett	Miele	NULL	NULL	NULL	...	6kg	NULL	37°C	Dryer
Lavamat	AEG	NULL	NULL	NULL	...	6kg	NULL	39°C	Dryer
....									

Families(Family,	Group)
Camcorder	Video
HomeVCR	Video
....	
Washer	HomeAppl
Dryer	HomeAppl

Groups(Group,	Area)
Video	ConsElectr
....	
HomeAppl	WhiteGoods

Fig. 5. Sample snowflake schema dimension tables for the product dimension

2.3 Summary and Conclusion

The *star / snowflake schema* approach allows modeling a wide range of simple multidimensional scenarios. From a performance point of view the *star schema* avoids a lot of lookup joins with the satellite tables but it is reprehensible from a schema design point of view.

Both traditional approaches however fail from an implementation and schema design point of view, if the *existence of dimensional attributes* depends on *values of the dimensional elements* in the classification hierarchy. As we have seen in the market research scenario, this problem becomes even worse, e.g. dimensional attributes '*Water*' and '*Load*' are only applicable for *washers* and not for *video equipment*.The Alternative Relational OLAP Approach

With our solution, we adopt the idea of [SmSm77] introducing a special class of attributes, called *discriminating attributes*. This kind of attributes holds relation names as their attribute values, which allows a real hierarchical representation of the problem of mapping a pyramid of concepts [Lore87], i.e. the classification hierarchy, to the relational data model.

In a first step, we can model each product group, i.e. leaf node of the classification tree, in a separate relation with all the node specific dimensional attributes. For example, the four sample product families from figure 1 are modeled in the following way:

Camcorder	ArticleID	Brand	VSys	BLT
	TR-75	Sony	HI8	2h
	TS-78	Sony	HI8	2h
	A200	JVC	N8	3h

Washer	ArticleID	Brand	Load	Water
	AB1043	Ariston	5kg	45l
	Princess	Miele	5kg	41l
	SuperII	Hoover	4kg	54l

HomeVCR	ArticleID	Brand	VSys	RC
	V-201	JVC	VHS	Yes
	ClassicI	Grundig	VHS-C	No

Dryer	ArticleID	Brand	Load	Temp
	Duett	Miele	6kg	37°C
	Lavamat	AEG	6kg	39°C

More formally, the schema of a classification node C at the lowest, i.e. first classification level with the classification attribute CA_1 is denoted by:

$$C(\underline{PA}, DA_1, ..., DA_m)$$

As usual, PA is the primary attribute for the join with the fact table and the set of DA_k ($1 \le k \le m$) denotes the dimensional attributes which are applicable to the classification node C.

The construction of the classification hierarchy is made in a bottom-up fashion, i.e. sets of classification nodes are grouped into a new high-level term, i.e. a new classification node. Suppose, in the j-th step, the classification nodes $\{C^1, ..., C^q\}$ with the set of locally valid dimensional attributes $\{DA_1^i, ..., DA_{m_i}^i\}$ for each C^i corresponding to the classification attribute CA_{j-1} are subsumed by the higher level node C corresponding to the classification attribute CA_j. The set of valid dimensional attributes is achieved by intersecting all attribute sets of the subsumed nodes.

$$\{DA_1, ..., DA_m\} := \bigcap_{i=1}^{q} \left\{ DA_1^i, ..., DA_{m_i}^i \right\}$$

For example, *Camcorders* and *HomeVCR* are classified into the class *Video*. *Washers* and *Dryers* are subsumed by the new higher level classification node *Home Appliances*. Furthermore, only those dimensional attributes are propagated to the new parent node, which are still valid there. Hence, the specific dimensional attributes '*BLT*' (for *Camcorder*) and '*RC*' (for *HomeVCR*) are lost, whereas the attributes '*VSys*' and '*Brand*' are propagated to the *video* class.

Generally, the schema of a new classification node C for the classification attribute CA_j ($1 < j \le p$) is algorithmically determined by:

$$C(\underline{PA}, DA_1, ..., DA_m, CA_1, ..., CA_{j-1}, CA_j) = \bigcup_{i=1}^{q} \pi_{(\underline{PA}, DA_1, ..., DA_m, CA_1, ..., CA_{j-1}, 'C^i')} C^i$$

The key point of this technique is that *each classification node* C^i *is added as a constant value for the new classification attribute* CA_j *of the new classification node* C.

In building the higher level classes, we intensively use the view mechanism of the relational database system at the implementation side. Below, the view definitions of the

product groups *Video* and *Home Appliances* are illustrated. In analogy to the formal description, each relation name appears as a constant value in the new view:

```
create view Video
  (ArticleID, Brand, VSys, Family) as
select ArticleID, Brand, VSys, 'Camcorder'
from Camcorder
union
select ArticleID, Brand,  VSys, 'HomeVCR'
from HomeVCR
```

Video(ArticleID,	Brand,	VSys,	Family)
TR-75	Sony	HI8	Camcorder
TS-78	Sony	HI8	Camcorder
A200	JVC	N8	Camcorder
V-201	JVC	VHS	HomeVCR
ClassicI	Grundig	VHS-C	HomeVCR

```
create view HomeAppl
  (ArticleID, Brand, Load, Family) as
select ArticleID, Brand, Load, 'Washer'
from Washer
union
select ArticleID, Brand, Load, 'Dryer'
from Dryer
```

HomeAppl(ArticleID,	Brand,	Load,	Family)
AB1043	Ariston	5kg	Washer
Princess	Miele	5kg	Washer
SuperII	Hoover	4kg	Washer
Duett	Miele	6kg	Dryer
Lavamat	AEG	6kg	Dryer

Each view holds the primary attribute, the applicable dimensional attributes and (in analogy to the star schema) all classification attributes. Furthermore, each view builds the basis for defining higher level classification nodes. This recursivly definition is shown below for the classification hierarchy depicted in figure 1.

```
create view ConsElectr
  (ArticleID, Brand, Family, Group) as
select ArticleID, Brand, Family, 'Video'
from Video
union ....

create view WhiteGoods
  (ArticleID, Brand, Family, Group) as
select
ArticleID, Brand, Family, 'HomeAppl'
from HomeAppl
union ....
```

```
create view Articles
  (ArticleID, Brand, Family, Group, Area) as
select ArticleID, Brand, Family, 'ConsElectr'
from ConsElectr
union
....
union
select ArticleID, Brand, Family, Group, 'WhiteGoods'
from WhiteGoods
```

In comparison to the traditional modeling approach (figure 4), only those attributes are available in the dimension table which are applicable for all dimensional elements. To address specific features, the corresponding dimensional sub-tables have to be used, whose names are specified as instances of the classification attributes. Figure 6 summarizes the modeling of the classification hierarchy using the proceeding, illustrated in this section. In analogy to figure 5 (snowflake schema), our approach can be straightforwardly normalized, too.

Articles(ArticleID,	Brand,	Family,	Group,	Area)
TR-75	Sony	Camcorder	Video	ConsElectr
TS-78	Sony	Camcorder	Video	ConsElectr
A200	JVC	Camcorder	Video	ConsElectr
V-201	JVC	HomeVCR	Video	ConsElectr
ClassicI	Grundig	HomeVCR	Video	ConsElectr
...				
AB1043	Ariston	Washer	HomeAppl	WhiteGoods
Princess	Miele	Washer	HomeAppl	WhiteGoods
SuperII	Hoover	Washer	HomeAppl	WhiteGoods
Duett	Miele	Dryer	HomeAppl	WhiteGoods
Lavamat	AEG	Dryer	HomeAppl	WhiteGoods

Fig. 6. Sample dimension table for the product dimension (alternative approach)

To put it into a nutshell, in the case of the traditional approach, the (senseless) query asking for total sales of *Home Appliances* by '*video system*' would result in a table scan of the dimension table, resulting in an empty join partner for the fact table and at last in a numerical zero. In our alternative approach, the query would be rejected, since *Home Appliances* does not contain a dimensional attribute '*video system*'.

3 Object-Relational Design

Another alternative and novel approach to overcome the limitations of the traditional star-/snowflake schema pattern is to use object-relational techniques. Since object-relational concepts are supported only to a certain degree and implemented in a very system-specific manner, we propose an object-relational schema design based on the capabilities of the IBM DB2/UDB V6.1 database system.

The design of an object-relational schema in DB2 is divided into two phases. In a first step, we have to define the type hierarchy and references based on types. Referring to these types, we are then able to 'instantiate' regular tables (also called: typed tables). Moreover, in opposite to the approach shown in the previous section, we have to proceed top-down when defining the object-relational schema of the dimensional structures.

3.1 Type Definitions

The super-type of a dimension holds only the most generic dimensional attributes and all possible classification attributes. For the ongoing example of the product dimension, the following DDL statement introduces the generic type of *Articles_T*, where the single articles are identified by the *ArticleID* attribute.

```
create type Articles_T as (brandvarchar(30),
           area   varchar(30),
           group  varchar(30),
           family varchar(30) );
```

Analogous to the classical concept of inheritance, special classes of products are derived from the more general classes of products and specific dimensional attributes are added to the derived type. Below are the SQL statements to define the necessary subtypes of the product classification hierarchy. For each sub-type, the keyword UNDER denotes the corresponding super-type.

```
create type ConsElectr_T under Articles_T as ( ... );
create type WhiteGoods_T under Articles_T as ( ... );
create type Video_T      under ConsElectr_T as(vidsysvarchar(30) );
create type HomeAppl_T   under WhiteGoods_T as(loadvarchar(30) );
create type HomeVCR_T    under Video_T as    (RC    char(1) );
create type Camcorder_T  under Video_T as    (BLT   varchar(5) );
create type Washer_T     under HomeAppl_T as (Water varchar(5) );
create type Dryer_T      under HomeAppl_T as (Temp  short) ;
```

Once we have defined the types of the dimensional structures, e.g. for the *Products* dimension and the *Shops* dimension, we are able to create a type for the fact table. Although this is not a mandatory step to design the schema of the fact table using OR technology, we are able to achieve some advantage when querying the database later. Therefore, the type of a fact table consists of two references to the super-types of the participating dimensions.

```
create type Facts_T as (
        ArticleID REF(Articles_T),
        ShopID REF(Shops_T),
        Period date,
```

```
        Sales  integer,
        Stock  integer,
        Price  integer);
```

It is worth mentioning here that this construction yields the following advantage: Consider the situation where we have to setup a data mart for a specific product group, e.g. video equipment. We are now able to create a type of a customized fact table allowing only references to the articles belonging to the video class. Therefore, we would substitute the reference to the generic articles type by a reference to the more specific type of *Video_T (... ArticleID REF(Video_T), ...)*. This design ensures already at the schema level (!!) that the fact table for this specific data mart will never contain a product other than a video article.

3.2 Typed Table Definitions

Once we have defined the type hierarchy for the classification schema, we are now in the position to 'instantiate' these types resulting in so-called 'typed tables'. Analogous to the type definition, we proceed top-down. For instantiation of the super-type, we need to introduce a object identifier attribute. For the ongoing example, we use the *ArticleID* as reference attribute (or primary key attribute in relational terminology), which content is given as user generated (in opposite to system generated). All dependent sub-tables are instantiated according to their type and referring to their direct super-type.

```
create table Articles of Articles_T (REF IS ArticleID user generated);
create table ConsElectr of ConsElectr_T under Articles inherit select privileges;
create table WhiteGoods of WhiteGoods_T under Articles inherit select privileges;
create table Video of Video_T under ConsElectr inherit select privileges;
create table HomeAppl of HomeAppl_T under WhiteGoods inherit select privileges;
create table HomeVCR of HomeVCRF_T under Video inherit select privileges;
create table Camcorder of Camcorder_T under Video inherit select privileges;
create table Washer of Washer_T under HomeAppl inherit select privileges;
create table Dryer of Dryer_T under HomeAppl inherit select privileges;
```

Again it is worth mentioning that this construction provides a huge advantage when dealing with changing dimensions. Consider again the product dimension. New articles may be added, some articles may be re-classified, and other articles may be deleted because they are no longer sold or their sales are no longer monitored. Since, however, all articles are of the same type, we can simply instantiate a specific type multiple times. Each resulting table reflects then a valid state of the classification hierarchy an can be used to analyze the fact data under certain valid time perspectives.

After creating the dimensional hierarchies, we can instantiate the fact table from the type *Facts_T*. Two aspects have to be considered explicitly. First of all, each fact gets a system generated object identifier *FactID*. More important is the instantiation of the references defined in the type specification. Each reference is pointing to an appropriate table, e.g. *ArticleID* is referencing the *Articles* table, not longer the *Articles_T* type.

```
create table Facts of Facts_T (ref is FactID system generated,
               ArticleID with options scope Articles,
               ShopID with options scope Shops);
```

3.3 Data Manipulation

The object-relational design of a data warehouse database has also some impact on data manipulation. Consider a simple insert statement for the product dimension. The only difference to a 'pure relational' insert is the casting of the article identification to the specific object identifier. Below is an example for a camcorder and a *Dryer* article.

```
insert into Camcorders ( ArticleID, brand, vidsys, blt, family, group, area) values
(Camcorders_T('TR-75'), 'Sony', 'HI8', '2h', 'Camcorder', 'Video', 'ConsElectr');
insert into Dryers ( ArticleID, brand, load, temp, family, group, area) values
(Dryers_T('Duett'), 'Miele', '6kg', '37oC', 'Dryer', 'HomeAppl', 'WhiteGoods');
```

To keep the consistency of references, we are now able to insert facts into the fact table. Again, we have to cast the values of the references to their corresponding type of the dimension tables, i.e. article identifiers are casted to *Articles_T*, shop names are casted to *Shops_T*.

```
insert into Facts (FactID, ArticleID, ShopID, Period, Sales, Stock, Price) values
(Facts_T('1'), Article_T('TR-75'), Shops_T('TeVi'), '1999-12-02', 45, 22, 998);
```

When querying the database we can take advantage of the references which may be visualized as predefined join paths. For example, grouping fact data by region (from the Shops table) and Products groups (from the Articles table) may be specified using the '->' operator without any explicit join.

```
select f.ShopID->region, f.ArticleID->group, SUM(sales)
   from Facts f
   group by f.ShopID->region, f.ArticleID->group
```

However, we can not retrieve specific attributes using this construction. The grouping by *Video Systems* for all video equipment would be expressed with the '*Video*' as the correct dimension table as follows:

```
select f.ShopID->region, a.vidsys, SUM(sales)
   from Facts f, Articles a
   where f.ArticleID = a.ArticleID
   group by f.ShopID->region, a.vidsys
```

In summary, object-relational technology provides a powerful tool for schema design in data warehouse environments. Especially, the design of a real classification hierarchy with classification and dimensional attributes finds an adequately representation using inheritance mechanisms on types and typed tables. Extending the object-relational representation to the fact table enables the designer to define some kind of structural constraints already at the schema level.

4 Summary and Conclusion

This paper addresses the problem of locally valid dimensional attributes within a classification hierarchy in the context of a multidimensional schema design. We show that the traditional way of a relational representation is not feasible and give two practicable solutions to this problem. The basic idea of the first proposed mechanism is based on the article of [SmSm77]. Building a pyramid of concepts in a bottom up manner using regular relational views may be seen as a seamless extension of the traditional *star /*

snowflake schema approach. The second top-down approach is based on object-relational techniques. We demonstrate how we can take advantage of object-relational concepts like types, typed subtables, inheritance on types and tables, and references. Again this method results in a flexible schema design.

References

ALTK97 Albrecht, J.; Lehner, W.; Teschke, M.; Kirsche, T.: Building a Real Data Warehouse for Market Research, to appear in: *8th International Conference and Workshop on Database and Expert System Applications* (DEXA'97, Sept. 1-5, 1997)

CoCS93 Codd, E.F.: Codd, S.B.; Salley, C.T.: *Providing OLAP (On-line Analytical Processing) to User Analysts: An IT Mandate*, White Paper, Arbor Software Corporation, 1993

GBLP96 Gray, J.; Bosworth A.; Layman A.; Pirahesh, H.: Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Total, in: *12th IEEE International Conference on Data Engineering* (ICDE´96, New Orleans, Louisiana, Feb. 26 -Mar. 1, 1996), pp. 152-159

GuMu95 Gupta, A.; Mumick, I.: Maintenance of Materialized Views: Problems, Techniques, and Applications, in: *IEEE Data Engineering Bulletin, Special Issue on Materialized Views & Data Warehousing* 18(1995)2, pp. 3-18

Hype99 N.N.: *Hyperion Essbase OLAP Server*, Hyperion Software Corporation, 1999 (http://www.hyperion.com/essbaseolap.cfm)

Inmo96 Inmon, W.H.: *Building the Data Warehouse*, 2nd edition, New York, John Wiley & Sons, 1996

Kimb96 Kimball, R.: *The Data Warehouse Toolkit*, New York, John Wiley & Sons, 1996

Lore87 Lorenzen, P.; *Constructive philosophy*, Amherst, Univ. of Massachusetts Press, 1987

Micr95 N.N.: *Microstrategy 6*, MicroStrategy Inc., 1999 (http://www.microstrategy.com/Products/index.asp)

Shos97 Shoshani, A.: OLAP and Statistical Databases: Similarities and Differences, in: *Proceedings of the 16th Symposium on Principles of Database Systems* (PODS'97, Tuscon, Arizona, May 12-14, 1997), pp. 185-196

SmSm77 Smith, J.M.; Smith, D.C.P.: Database Abstractions: Aggregation and Generalization, *ACM Transactions on Database Systems 2(1977)2*, pp. 105-133

SQL99 N.N.: *ISO/IEC 9075*: 1999, Informatik Technology - Database Languages - SQL, 1999

VaSe99 Vassiliadis, P.; Sellis, T.: A Survey of Logical Models for OLAP Databases. In: *SIGMOD Record 28(1999)4*

Functional Dependencies in Controlling Sparsity of OLAP Cubes

Tapio Niemi[1], Jyrki Nummenmaa[1], and Peter Thanisch[2]

[1] Department of Computer and Information Sciences, University of Tampere
FIN-33014 University of Tampere, Finland
{tapio, jyrki}@cs.uta.fi
[2] Institute for Computing System Architecture,
University of Edinburgh, Edinburgh EH9 3J2, Scotland
pt@dcs.ed.ac.uk

Abstract. We will study how relational dependency information can be applied to OLAP cube design. We use dependency information to control sparsity, since functional dependencies between dimensions clearly increase sparsity. Our method helps the user in finding dimensions and hierarchies, identifying sparsity risks, and finally changing the design in order to get a more suitable result. Sparse raw data, a large amount of pre-calculated aggregations, and many dimensions may expand the need of the storage space so rapidly that the problem cannot be solved by increasing the capacity of the system. We give two methods to construct suitable OLAP cubes. In the synthesis method, attributes are divided into equivalence classes according to dependencies in which they participate. Each equivalence class may form a dimension. The decomposition method is applied when candidates for dimensions exist. We decompose dimensions based on conflicts, and construct new cubes for removed dimensions until no conflicts between dimensions exist.

1. Introduction

On-Line Analytical Processing (OLAP) has gained popularity as a method to support decision making in situations where large amounts of raw data should be analysed. In OLAP queries are made against structures called OLAP cubes. The design of a cube is based on knowledge of the application area and the types of queries the users are expected to pose.

In practice, the users may want to speculate about the effects of, e.g. changing the way their company has arranged its organisation. Also, there may be needs to analyse data in ways, which were not anticipated at the time when the OLAP cubes were designed. It is common practice for some OLAP users to frequently reorganise their OLAP cubes even though the nature of the underlying raw data has not changed.

Although there are various different ways in which the users may want to arrange the OLAP cube, there are also significant differences between different designs in terms of efficiency and practicality as some design can lead to. For example, a cube can be extremely sparse, in a sense that many of the data items are missing or zero because of the nature of the data, or some other designs can be problematic for query formulation (incorrect aggregations).

Y. Kambayashi, M. Mohania, and A M. Tjoa (Eds.): DaWaK 2000, LNCS 1874, pp. 199–209, 2000.
© Springer-Verlag Berlin Heidelberg 2000

However, although there is some research done about the design goals for OLAP cubes, this research can be seen as far from being complete and many important design issues still require further attention. To date, the research has concentrated in physical design such as indexing and clustering. There is also some work done on the conceptual level description of OLAP applications using e.g. ER diagrams [2].

We divide OLAP cube design into three phases: conceptual modelling, logical design, and physical design. The borders of these are not very clear, and they are different from traditional database design.

By conceptual modelling we mean describing and analysing the concepts and their relationships. The result of conceptual modelling is a conceptual schema, which explicitly present concepts and their relationships. The starting point of conceptual modelling varies. A situation is often that we have a pre-existing data warehouse. Other possibilities are that we only have an operational database ready, and we can even start designing the cube first and then design and implement the data warehouse for that cube. Logical design starts from the conceptual schema and its result will be an OLAP schema, i.e. cubes with dimensions and their hierarchies. Unlike traditional database design, designing of logical database schema, e.g. relational database schema [3], is not included in logical OLAP design but physical design. The aim of physical design is to find an efficient implementation for the desired data cube. The result can be a relational star (snow flake) schema or a multidimensional schema. Physical design also contains optimisation of the chosen storage structure, for example building indexes.

This work is organised as follows. In Section 2 we study logical design and its aims in more detail. Section 3 is a review of related work. In Section 4, we study different properties of dependency sets and their effects on sparsity. At the end of the section we give two different approaches to construct OLAP schemas avoiding sparsity. Finally, the conclusions are presented in Section 5.

2. Logical Design

The aim of logical design is to produce a good schema for an OLAP cube or a set of cubes. The high level aims are to produce a cube in which queries can be constructed easily, answered correctly, and evaluated efficiently. The first aim means that the system should be easy to use, and the user should be able to construct complex queries easily. By correctness we mean that the user gets a correct and relevant answer to the query. Efficiency means that the query can be evaluated quickly using minimal amount of resources.

The starting point of logical design is a formal conceptual schema of a data warehouse. The conceptual schema is supposed to be complete, i.e. it describes all information stored in the data warehouse. The conceptual schema contains concepts and relationships between them. In logical design, we operate with concepts and relationships in order to get a schema that represents the logical model of an OLAP cube. We can derive new concepts and relationships according to particular rules, and also remove concepts and relationships if the rules allow it. A basic type of these rules are the inference rules of the relational dependency theory [3, 4].

The result of logical design can be evaluated by comparing it to some normal forms. The normal forms will be something like relational normal forms: a particular normal form has some desirable properties related to the correctness of aggregation,

sparsity, etc. On logical level our high level aims are completeness, summarizability, and controlling sparsity.

Controlling Sparsity. Sparsity means that an OLAP cube contains empty cells. A simple measure for sparsity is the ratio of empty cells compared to all cells. The ideal case is that sparsity is zero but this is seldom possible to achieve in real applications. Pendse has noticed that sparsity can increase more than exponentially in situations containing many pre-calculated aggregation values [12]. This kind of database explosion cannot be avoided using efficient storage methods for sparse data, because the real amount of data increases due to pre-calculated aggregation information. The only solution is to ensure that the cube, which stores the raw data, is not sparse.

Completeness. Cabibbo and Torlone [1] mean by completeness that all information stored in operational databases is possible to achieve using the OLAP cube. Completeness in this sense is related to the data warehouse design rather than logical OLAP design. In OLAP, we can form the meaning of completeness in a different way: it should be possible to access any information stored in the data warehouse using an OLAP cube or cubes.

Summarizability. Lenz and Shoshani have studied summarizability in OLAP [6]. They give three necessary conditions for summarizability, and they also assume that these conditions are sufficient. The conditions are:

1. disjointness of category attributes,
2. completeness, and
3. correct use of measure (summary) attributes with statistical functions.

Disjointness requires that attributes in dimensions form disjoint subsets over the elements. Completeness means that all elements exist and every element is assigned to some category in the level above it in the hierarchy. Correct use of measure attributes with statistical functions is the most complex of these requirements, since it depends on the type of the attribute and the type of the statistical function. Measure attributes can be classified into three different types, which are flow, stock, and value-per-unit. Knowing the type of an attribute we can conclude which statistical functions can be applied to the attribute.

3. Approaches to Logical OLAP design

The logical cube design is not a widely studied area. Most of existing works are related to physical design, i.e. they study methods to implement a cube efficiently. In this section we study some papers related to logical OLAP design.

Pedersen and Jensen [11] discuss requirements for multidimensional data models. Their nine requirements are:

1) explicit hierarchies in dimensions,
2) symmetric treatment of dimensions and measures,
3) support for multiple hierarchies in a dimension,
4) correct aggregation of data,
5) support for non-strict hierarchies,
6) handling of many-to-many relationships between facts and dimensions,
7) change of data over time should be supported,
8) uncertainty could be associated to data, and
9) data with different levels of granularity should be allowed.

Explicit hierarchies in dimensions means that hierarchies should be explicitly shown in the schema to help user's understanding of hierarchical structures in dimensions. Symmetric treatment of dimensions and measures means that it should be possible to make a measure as a dimension and visa versa. It can sometimes be useful to group items according to an attribute usually used as a measure; e.g. to group products according to the profit they give. By multiple hierarchies in a dimension the authors mean that a dimension hierarchy is allowed to be an acyclic graph, that is, there can be several paths from the top level to the most detailed level. For example, the time dimension can have hierarchies like day-week, and day-month-year.

The fourth item, correct aggregation of data, ensures that aggregated values are not misleading. The idea of the fifth item, support for non-strict hierarchies, is to allow many to many relationships between different levels in a dimension. Many to many relationships between facts and dimensions means that there can be many measure values per one combination of dimension values, i.e. the measure should be a set of values instead of one value. The situation can often be avoided by increasing dimensions but this can sometimes be unnatural. Time and changes are important aspects in OLAP, since OLAP is often used in analysing historic data. Time itself is usually treated as a dimension, but changes are more difficult, for example hierarchies can change in time. The eighth item means that the system should be able to handle uncertain or fuzzy information. Finally, the last item, data with different levels of granularity should be allowed, means that it should be possible to store data in different levels of hierarchy, not only to the lowest level. However, this easily causes problems in aggregations.

The authors propose a model providing all nine properties. The model is an algebraic one, which may reduce its usability, but it has a large expressive power.

Li and Wang [8] present a model for multidimensional databases. A database can contain several data cubes. They also define an extension for relational algebra, called grouping algebra, which can be used in OLAP queries.

In some works the ER model [2] is applied to the cube design. Sapia et al. [13] extend the ER model by adding three primitives into it in order to allow expressing the multidimensional structure of data: a fact relationship set, a dimension level set, and a classification set. However, the method does not offer much help for the design process but the designer has to know what kind of structure is suitable for the current application. Further, the model does not have features to help to achieve desirable properties of the OLAP cube (summarizability, efficiency).

Golfarelli et al. [5] present a semi-automatic modelling method based on an existing ER schema. The method is called Dimensional Fact model. It can automatically find possible dimension hierarchies and allows the user to modify them by adding or removing levels. Summarizability is also noticed; measure attributes are classified additive, semi-additive, and non-additive. Measures are supposed to be atomic. If many measures are needed, several cubes can be constructed. Two cubes are defined to be compatible if they have at least one common dimension. To ensure completeness in aggregations, many to one mapping has to be hold between levels in a dimension. The algorithm for building dimensions may help to get orthogonal dimensions, but this is not studied in the paper. However, some relationships may be lost while construction dimensions, and they may cause problems with orthogonality.

A method by Cabibbo and Torlone [1] is based a multidimensional data model, called MD, which main parts are a dimension and a fact table. The method has some

similarities to the method of Golfarelli et al. They both start from the existing ER schema of an operational database. The method by Cabino and Torlone has four steps:

1. identification of measures and dimensions,
2. restructuring the pre-existing ER schema,
3. derivation of the dimensional graph, and
4. translation into the MD model.

The first two steps can be parallel and they are often iterated. The dimensional graph is used to represent measures and dimensions. It can be constructed from the restructured ER schema, and the dimensions can be easily derived from the dimensional graph. The MD model does not explicitly offer help in designing OLAP cubes with good properties for aggregation correctness and controlling sparsity.

Lehner et al. [7] present normal forms for multidimensional databases. The normal forms are designed to help in ensuring summarizability and controlling sparsity. The normal form requires that there exists a functional dependency between the hierarchical levels in a dimension, but no functional dependency is allowed between different dimensions. The normal form presented helps to avoid problems with incorrect aggregation and sparsity but the normal form itself may sometimes be impossible to achieve in practise, because totally independent dimensions are not always possible to find in many applications. The orthogonality of dimensions is studied only with respect to (unary) functional dependencies, and summarizability is not understood in as large a sense as in the paper by Lenz and Shoshani [6].

Wang and Li [14] discuss the use of orthogonality in optimising search for summary data. Although the aim of the paper is to improve efficiency of query processing, it discusses some topics related to logical design, such as the relationship between multivalued dependencies and orthogonality: if there is a multivalued dependency between dimensions, the dimensions are orthogonal.

4. Applying Dependency Information in OLAP Design

In this section we show how functional dependencies [4] can be applied to logical OLAP design. We assume that the reader is familiar with some dependency theory.

4.1 Properties of Dependency Sets

In this subsection we study dependency sets and their properties in the spirit of Nummenmaa [10]. Nummenmaa has noticed that certain type of dependency sets lead to bad relational database schemas. In what follows we show that these results can also be applied in OLAP design. We study how certain properties of a dependency set reflect to properties of the resulting OLAP cube. We concentrate only in functional dependencies (FDs). The set of FDs is denoted by F, and the set of left hand sides of F by LHS(F). The dependency set is called unary, if it contains only FDs with a singleton left hand side. Unary dependency sets are easier to handle than non-unary sets, because of simple inference rules, but expressive power decreases.

We will need the following definitions. If $X \rightarrow Y$ and there is no such $X' \subset X$ that $X' \rightarrow Y$, then $X \rightarrow Y$ is a full dependency, and we write $X \mapsto Y$. We say that $X^+$is a closure of X, i.e. X^+ contains all attributes that X determines directly or transitively. We write $X^+_f = \{A \mid X \mapsto A\}$, $X^{\rightarrow} = X^+ - X$, and $X^{\mapsto} = X^+_f - X$.

RHS-intersection free. A dependency set is RHS-intersection free, if for any X, Y ∈ LHS(F), whenever $X^{\mapsto} \cap Y^{\mapsto} \neq \emptyset$, then either X→Y or Y→X. [10]
An RHS-intersecting dependency set can still be unary thus RHS-intersection free property is a natural extra demand for unary dependency sets.
Split free. We say that X splits Y if $X^{\rightarrow} \cap Y \neq \emptyset$ and not X→Y. A set of FDs is split free if no X∈LHS(F) splits any Y∈LHS(F). [10]
Every unary dependency set is also split free. Splitting often exists together with RHS-intersecting property.

4.2 Estimating Sparsity

By *minimum sparsity* we mean the theoretical minimum amount of sparsity in a cube. We usually have some design constraints, like FDs, for logical cube design. As an example, we have a cube with three dimensions: employee, customer and product. The measure is the amount of sold products. With no FDs (or other constraints), the minimum sparsity is zero. However, if we know that an employee sells only one product, we get a FD employee→product. Now, the cube is necessarily sparse, because most of product-employee combinations will have an empty value. If we have 10 employees, 20 customers and 5 products, then the size of the cube is 1000 cells. We can slice the cube according to employees and get 10 sub cubes. Every sub cube can have at most 20 non-empty cells, since an employee can sell only one product but maybe to all customers. Further, the number of non-empty cells is 10x20=200, while the total number of cells is 1000, thus the sparsity in the cube cannot be less than 80%. Next we study how different properties of dependency sets effect sparsity.

Unary Dependency Sets. Let C(U) be a data cube schema, U a non-empty set of attributes forming the dimensions and D a set of unary FDs. Let X=U, and for all such FDs A→B ∈D that B∈X, remove B from X. Let c be an attribute. The notation |c| means the amount of different values of c. The minimum sparsity of the cube is now

$$1 - \prod_{x \in X} |x| \Big/ \prod_{c \in U} |c| .$$

If the dependency is A→B and B→A (a key dependency), A or B can be removed but not both of them. A key dependency is worse for sparsity than a normal FD. It leads to even sparser cubes, as values are spread diagonally (Figure 1).

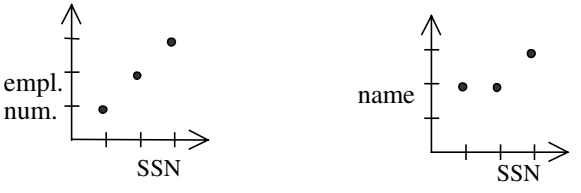


Fig. 1. Two cubes with different dependencies:
i) SSN→ empl. num and empl. num →SSN, ii) SSN→name

RHS-intersecting Dependency Sets. If a dependency set is not RHS-intersection free, it is not possible to construct one non-sparse OLAP cube because of a

RHS-intersection(s) between some dimensions. The following example illustrates this. We use a schema drawing technique simplified from the one presented by Niemi et al. [9]. Text boxes represent attributes, and arrows functional dependencies. A two head arrow means a symmetric functional dependency, i.e. a key dependency.

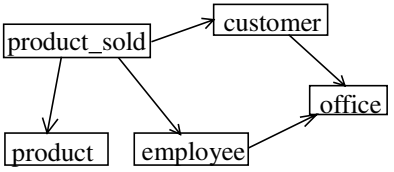


Fig. 2. Employee and customer use only one office

In the schema above, employee and customer do business only in one office which is always the same for both of them. If the number of product items sold is the measure, possible dimensions are: product, employee, customer, and office. Further, the dependencies are: product_sold→customer employee product office, customer→office, and employee→office. A natural choice for dimensions is product, employee, and customer. A problem in the dependency set is that both customer and employee determine office but there is neither dependency customer→employee nor employee→customer, i.e. the dependency set is RHS-intersecting. From this it follows that only such employee-customer pairs that use the same office can have a non-empty measure. The situation remains, if we choose the office as a dimension. In this case, the minimum sparsity can be calculated using a formula 1-((|customer| |employee| |product|) / (|customer| |employee| |product| |office|)).

Non-unary FDs. In this subsection we study non-unary dependency sets and their properties.

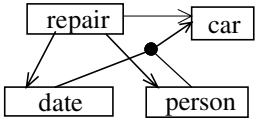


Fig. 3. date person → car

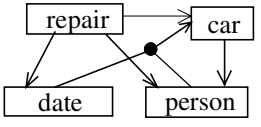


Fig. 4. date person→car, car→person

In Figure 3, dependencies date person→car and repair→car make the dependency set RHS-intersecting. For this reason the cube will be sparse if we choose date, person, and car as dimensions, because a car can be repaired only once per one date-person combination. We can extend the calculation rule of minimum sparsity to non-unary FDs simply allowing the left hand side to contain attributes from several dimensions. In Figure 3, the minimum sparsity will be 1-((|date| |person|) / (|date| |person| |car|)). We could get a better cube if only date and person were the dimensions but then it would be difficult to analyse the repairing history of a particular car. To avoid the problem we can construct multiple cubes with dimensions: car, date; car, person; and data, person.

In Figure 4 we have a situation where the car splits the set {date, person}, since the $\{car\}^+ \cap \{date, person\} \neq \emptyset$ but the car does not determine {date, person}. In practice this means, that the same person always brings the car to the garage, and a person can

bring only one car a day to the garage. If car, date, and person are the dimensions, the sparsity will be 1-((|date| | person|) / (|date| |person| |car|)) that is the same as without the car→person dependency. However, the car→person dependency slightly increase the sparsity by demanding that there has to be at least as many cars as persons.

Finally, the last case is that the left hand sides of dependencies intersect. An example on this is seen in Figure 5. However, the example schema is also RHS-intersecting. The LHS intersection is caused by the date attribute being a part of the left hand side for both owner data→discount, and person date→car dependencies. LHS-intersection does not seem to be very problematic for OLAP cubes, because these situations are usually RHS-intersecting, too. Therefore we do not study LHS-intersection more detailed in this work.

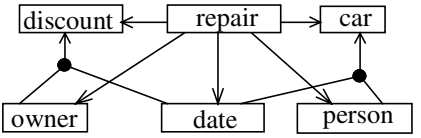


Fig. 5. date person→car, date owner→discount

Normal Forms. According to the properties of the dependency sets studied above, we can define normal forms to avoid those problematic situations. Relational normal forms, except the first one, are hardly suitable for OLAP cubes, because of different aims in operational and OLAP databases. In general, the aims of OLAP normal forms are to ensure both minimal amount of sparsity and correctness of aggregations, but we concentrate mostly to controlling sparsity in this work. OLAP normal forms can be classified into three categories. Multidimensional normal forms are related to dependencies between dimensions, dimensional normal forms concern intra-dimensional dependencies, and first normal form dependencies between the dimension set and the measure attribute.

Following Lehner et al. [7], a data cube is in:

- dimensional normal form if there is a FD from each level to the next level in a hierarchy.
- multidimensional normal form if it is in dimensional normal form and there are no FDs between dimensions.

The aim of the dimensional normal form is to guarantee completeness in aggregations, while the multidimensional is mostly used in controlling sparsity of the OLAP cubes.

We define two new normal forms for a multidimensional OLAP cubes. The normal forms are based on the properties of functional dependency sets. We say that an attribute participates in a dependency if it exists on the left or the right hand side of the dependency. A data cube is in:

- RHS intersection normal form if it is in multidimensional normal form and there are no RHS-intersecting dependencies between dimensions.
- Split free normal form if it is in RHS intersection normal form and there are no $X \in LHS(F)$ that splits any $Y \in LHS(Y)$ that belongs to a different dimension.

4.3 Producing Normalised OLAP Schemas

We can recognise two different situations in logical OLAP design. In the first one we know the dimensions, and want to decompose them based on FDs. In the second one we only have a conceptual schema with the dependency information and we want to build the dimensions. The first case is analogical to relational decomposition and the second one to synthesis. In decomposition we decompose to reduce conflicts, and in the synthesis we combine compatible attributes in the same dimension. It is typical for decomposition that it results in too many dimensions (loss of information and need to combine data from several dimensions) and synthesis in too few (sparsity).

Synthesis. We construct equivalence classes of attributes based on functional dependencies they participate. The measure attribute cannot be taken into account, because it usually functionally determines all other attributes. Let U be a set of attributes, M the measure concept (the concept whose some attributes are the actual values to measure), S=U-{M}, and F the set of functional dependencies of attributes in S. We form an undirected dependency graph for S as follows. All attributes of S will be nodes. Let A, B∈S. There is an edge between A and B if and only if there is such a FD X→Y ∈ F that A∈X and B∈Y, or A∈Y and B∈X. Further, A and B belong into the same equivalence class, if there is a path in the dependency graph between A and B.

An equivalence class can form a dimension if it is in dimensional normal form [7]. The equivalence classes cannot be shared into different dimensions, because in that case we necessarily would have a dependency between dimensions. A dimension cannot contain attributes from several equivalence classes, because according to the dimensional normal form there has to be a functional dependency between the attributes in the same dimension.

Example 1. Figure 6 illustrates the algorithm. We want to measure the profit of sold products, i.e. the product_sold is the concept to measure and profit the actual value to measure. We can also notice that we have a FD from the product lot to the employee (for example an employee sells the products (s)he buys), thus the employee cannot be an individual dimension but it can be used to classify products, i.e. we can group products according to employee bought them.

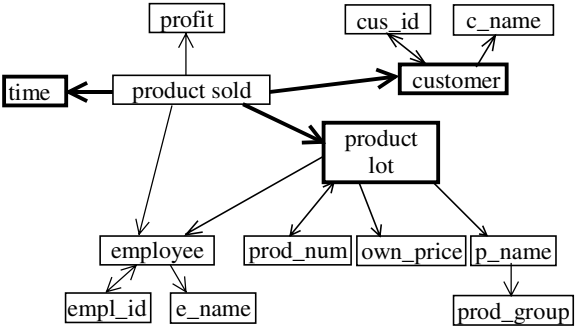


Fig. 6. OLAP schema of a company. Profit is the measure. Potential dimensions are time, customer, and product lot. Employee cannot be a dimension

The set S will be {time, employee, empl_id, e_name, customer, cus-id, c_name, product_lot, prod_num, own_price, p_name, prod_group}. The set D={employee •empl_id e_name, customer •cus-id c_name, product_lot •employee product_num own_price p_name, p_name •prod_group, cus_id •customer, empl_id •employee, prod_num •product}. We get three equivalence classes {time}, {employee, empl_id, e_name, product_lot, product_num, own_price, p_name, prod_group}, and {customer, cus_id, c_name}. All equivalence classes are in dimensional normal form, so they can form three dimensions.

Decomposition. We can use a decomposition technique when initial dimensions are given. In decomposition we construct multiple cubes to avoid dependencies between dimensions. We can divide a cube using dimensions or values of a dimension.

We assume that dimension candidates are in dimensional normal form. In the first case conflicting dimensions are removed until the desired normal form is achieved. A new cube is constructed from the removed dimensions and then the process is repeated for it. The resulting OLAP schema contains multiple cubes having zero as the minimum sparsity. Example 2 illustrates this.

The second possibility is to divide a cube according to some grouping attribute. For example for the schema presented in Figure 2, we can construct a separate cube for every office. However, this usually violates the completeness condition, since completeness can be understood so that if we have some attribute as a dimension, then it has to contain all values of that attribute. The two methods can also combined.

Example 2. Normalising the schema in Figure 2 gives the tables in Figure 7.

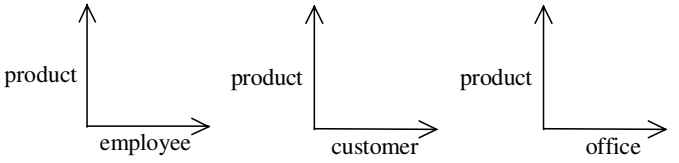


Fig. 7. Normalised OLAP schema

The minimum sparsity of the OLAP schema in Figure 7 is zero. We cannot form a non-sparse cube with employee and customer as dimensions (every employee cannot do business with every customer, if there are customers using different offices). The normalisation is not lossless; we lose a connection between employees and customers. However, if we allow some sparsity and also construct a cube with employee and customer as dimensions, no information will be lost and total sparsity is still much less than in a case of one four dimensional cube.

5. Conclusions

We have studied functional dependencies in the context of logical OLAP cube design. The OLAP cube should be suitable in two aspects: sparsity and correctness of aggregations. Using functional dependency information we can achieve better results in both of the aims.

OLAP schema should not contain any functional dependencies between different dimensions but there should be a functional dependency from each level to the higher level in a hierarchy [7]. Moreover, we have noticed that there should not exist such situations that attributes belonging into different dimensions determine some another and the same attribute or attributes together determines another attribute (non-unary FDs). Independences of dimensions guarantee the minimal amount of sparsity while functional dependencies between hierarchies ensure completeness in aggregations. Furthermore, to correct aggregations also the types and semantics of attributes has to be known to be able to know which statistical functions can be applied.

References

1. Cabibbo, L. and Torlone, R.: A logical approach to multidimensional databases, *6th International Conference on Extending Database Technology (EDBT'98, Valencia, Spain, 23-27 March)*, G. Alonso (eds.), LNCS, Vol. 1377, pages 183-197, Springer, 1998.
2. Chen, P. P.: The Entity-Relationship model: Toward a Unified View of Data, *ACM Transactions on Database Systems*, 1(1), 1976.
3. Codd, E. F.: A relational model for large shared data banks, *Communications of the ACM*, 13(6), 1970.
4. Codd, E. F.: Further Normalization of the Data Base Relational Model, *Data Base Systems*, Courant Computer Science Symposia Series 6, R. Rustin (ed.), Prentice-Hall, Englewood Cliffs, New Jersey, 1972.
5. Golfarelli, M., Maio, D., and Rizzi, S.: Conceptual design of data warehouses from E/R schemes, *Proceedings of the 31st Hawaii International Conference on System Sciences*, 1998.
6. Lenz, H.-J. and Shoshani, A.: Summarizability in OLAP and Statistical Data Bases, *Ninth International Conference On Scientific And Statistical Database Management (SSDBM)*, pages 132-143, 1997.
7. Lehner, W., Albrecht, J., and Wedekind, H.: Multidimensional normal forms, *Proceedings of the 10th International Conference on Scientific and Statistical Data Management (SSDBM'98)*, Capri, Italy, 1998.
8. Li, C. and Wang, X. S.: A data model for supporting on-line analytical processing, *Proceedings of the Conference on Information and Knowledge Management (CIKM'96)*, pages 81-88, 1996.
9. Niemi, T., Nummenmaa, J., and Thanisch, P.: Applying dependency theory to conceptual modelling, *Topics in Conceptual Analysis and Modeling*, Czech Academy of Sciences' Publishing House Filosofia, Prague, 2000. (to be published)
10. Nummenmaa, J.: Designing Desirable Database Schemas, Department of Computer Science, University of Tampere, A-1995-1, Tampere, 1995. (Ph.D. Thesis)
11. Pedersen, T. and Jensen, C.: Multidimensional data modeling for complex data, *Proceedings of the International Conference on Data Engineering (ICDE'99)*, 1999.
12. Pendse, N.: Database Explosion, *The OLAP Report*, 2000. Available at http://www.olapreport.com/DatabaseExplosion.htm.
13. Sapia, C., Blaschka, M., Höfling, G., and Dinter, B.: Extending the E/R model for the multidimensional paradigm, *Advances in Database Technologies*, Y. Kambayashi et al. (eds.), pages 105-116, LNCS 1552, Springer, 1998.
14. Wang, X. S. and Li, C.: Deriving orthogonality to optimize the search for summary data, *Information Systems*, 24(1), pages 47-65, 1999.

An OLAP-based Scalable Web Access Analysis Engine

Qiming Chen, Umeshwar Dayal, Meichun Hsu

HP Labs, Hewlett-Packard, 1501 Page Mill Road, MS 1U4, Palo Alto, CA 94303, USA
{qchen,dayal,mhsu}@hpl.hp.com

Abstract. Collecting and mining web log records (WLRs) from e-commerce web sites has become increasingly important for targeted marketing, promotions, and traffic analysis. In this paper, we describe a scalable data warehousing and OLAP-based engine for analyzing WLRs. We have to address several scalability and performance challenges in developing such a framework. Because an active web site may generate hundreds of millions of WLRs daily, we have to deal with huge data volumes and data flow rates. To support fine-grained analysis, e.g., individual users' access profiles, we end up with huge, sparse data cubes defined over very large-sized dimensions (there may be hundreds of thousands of visitors to the site and tens of thousands of pages). While OLAP servers store sparse cubes quite efficiently, rolling up a very large cube can take prohibitively long. We have applied several non-traditional approaches to deal with this problem, which allow us to speed up WLR analysis by 3 orders of magnitude. Our framework supports multilevel and multidimensional pattern extraction, analysis and feature ranking, and in addition to the typical OLAP operations, supports data mining operations such as extended multilevel and multidimensional association rules.

1 Introduction

Commercial web sites typically generate large volumes of web log records (WLRs) daily. These WLRs can be collected and mined to extract customer behavior patterns, which may then be used for a variety of business purposes such as making product recommendations, designing marketing campaigns, or redesigning the web site. Numerous commercial tools (e.g., WebTrends [13] and Net.Genesis [12]) are available for analyzing WLRs (and other data sources) and generating reports for business managers [3,7,11]. However, these tools typically provide only a fixed set of pre-configured reports, have limited on-line analytical capabilities, and do not support more sophisticated data mining operations such as customer profiling or association rules.

On-line analytical processing (OLAP) tools are designed to support complex, multi-dimensional and multi-level on-line analysis of large volumes of data stored in data warehouses [1,2-4,8,10]. In our prior work, we have described a scalable framework developed on top of an Oracle-8 based data warehouse and a commercially available multi-dimensional OLAP server, Oracle Express, which we have

Y. Kambayashi, M. Mohania, and A M. Tjoa (Eds.): DaWaK 2000, LNCS 1874, pp. 210–223, 2000.
© Springer-Verlag Berlin Heidelberg 2000

used to develop applications for analyzing customer calling patterns from telecom networks and shopping transactions from e-commerce sites [6]. In this paper, we describe a Web access analysis engine implemented on this framework to support the collection and mining of WLRs at the high data volumes typical of large commercial Web sites. In developing this engine, we encountered additional performance and functionality problems, which we address in this paper.

Typically, hundreds of millions of WLRs are created daily, and the architecture must support loading and processing rates that match the input rate. Further, since WLRs are continuously collected, it is important to mine them in "real-time" to dynamically detect trends and changes in traffic patterns, so as to dynamically make business decisions (e.g., recommending products, targeting advertisements, offering promotions). The results of the analysis are represented by summary information at multiple levels of granularity, e.g., hourly, daily, weekly, and monthly, as well as along multiple dimensions. These summaries must be stored and incrementally updated.

While a data warehouse/OLAP framework is capable of dealing with huge data volumes, it does not guarantee that the summarization and analysis operations can scale to keep up with the input data rates. Specifically, for Web access analysis, we want to introduce a number of fine-grained dimensions, resulting in very large, very sparse data cubes, which pose serious scalability and performance challenges to data aggregation and analysis, and more fundamentally, to the use of OLAP for such applications. For example, in one application, a newspaper Web site received 1.5 million hits a week against pages that contained articles on various subjects. We wanted to profile the behavior of visitors from each originating site at different times of the day, including their interest in particular subjects and which referring sites they were clicking through. We modeled the data using four dimensions: ip address of the originating site (48,128 values), referring site (10,432 values), subject uri (18,085 values), and hours of day (24 values). The resulting cube contained over 200 trillion cells! (Clearly, the cube was extremely sparse.) Each of the dimensions participated in a 2-level or 3-level hierarchy. We estimated that to rollup such a cube along these dimension hierarchies using the regular rollup operation supported by the OLAP server would require some 10,000 hours (i.e. more than one year), on a single Unix server! The problem is that while most MOLAP or ROLAP engines provide efficient mechanisms for caching and storing sparse cubes, they lack efficient mechanisms for rolling up such cubes.

We have introduced several "non-traditional" OLAP solutions to handle the above scalability issues for our Web access analysis engine. We develop approaches for dealing with very large, sparse cubes. We introduce the notion of *high-diagonal cubes* to replace the traditional "embedded-total" cubes (in which all intermediate summaries all the way to the top of each dimension hierarchy are computed at load time), and we use *direct binning*, instead of rollup, to populate the high-diagonal cubes. We further reduce the computation load by selecting *high-profile dimension elements*. These approaches allow us to speed up Web log analysis by three orders of magnitude. Additional bookkeeping is required to maintain the relationships be-

tween the high-level data cubes containing aggregates and the low-level data cubes containing detailed data, thus allowing users to drill-down selectively.

This data-warehouse/OLAP framework and Web access analysis engine have been implemented at HP Labs. Our experience has demonstrated that it is possible to overcome the performance problems of handling sparse data cubes, and to automate the whole operation chain, including data filtering, loading, incremental summarization and analysis. The application, including the optimizations described in this paper, was implemented by *OLAP programming* in the scripting language provided by the OLAP server. Further, as we described in [5,6], we use OLAP programming to compute various classes of multi-level and multi-dimensional patterns and association rules. In particular, we had introduced several families of association rules such as *scoped association rules* and *functional association rules*. We show how these classes of patterns and association rules can be used in analyzing WLRs, and we define a new class of *time-variant rules*, which are also useful for Web access analysis. Thus, we use the OLAP server as a computation engine to support data mining operations. In this respect, we are in agreement with the approach described in [9] that uses OLAP tools to support large-scale data mining.

Section 2 introduces the architecture of the OLAP-based Web access analysis engine and shows how it automates the operation chain. Section 3 discusses the scalability issues. Section 4 illustrates pattern and rule discovery for Web access analysis. Finally, Section 5 gives some conclusions and summarizes lessons learnt.

2 Data Warehouse/OLAP Based Web Access Analysis Engine

Almost all e-commerce applications are Web based. *Web log records* (WLRs) are generated to represent information specific to each Web access attempt. Each WLR typically contains, among other things, the IP address of origin site, the access time, the referring site, the URI of the target site (i.e., the Web page or object accessed), the browser method and protocol used.

There are two general tasks for Web access analysis: (1) compute multi-dimensional summary information from a number of raw WLRs; and (2) derive usage patterns and rules for supporting business intelligence. Below are some examples.

Usage Analysis. The volume and distribution of hits for specific topics, dimensioned by origin site, referring site and time at multiple levels, can be used as quantitative measures for personalizing the delivery of content to customers in different areas and at different times.

Web Site Traffic Analysis. The volume and distribution of hits for target sites, dimensioned by referring site and time, can be used for resource and network planning distributing workload over multiple sites, creating mirror sites, or caching content.

Business Rules Discovery. The change of access rates to a Web site can provide indications of changing customer interests and behavior. For instance, the correla-

tion between a content topic and certain origin sites in an area describes the interest of the customers in that area. While such relationships are helpful for making ma r-keting promotion decisions, the changes in such relationships may be even more significant, since such changes usually reflect real-time trends of changes in custom-ers' interest, reactions to a marketing campaign, as well as the impact of compet i-tors. To catch such relationships requires us to mine for association rules contin u-ously and incrementally.

We measure Web access in terms of volumes and probability distributions, which are expressed in the form of data cubes. A cube C has a set of underlying *dimensions* $D_1,\ldots,D_n$, and is used to represent a multidimensional *measure*. Each cell of the cube is identified by one element value from each of the dimensions, and contains a value of the measure. We say that the measure *is dimensioned by* $D_1,\ldots,D_n$. The set of elements of a dimension D, called the *domain* of D, may be limited (by the OLAP *limit* operation) to a subset. A sub-cube (slice or dice) can be derived from a cube C by dimensioning C by a subset of its dimensions, or by limiting the domains of its dimensions.

For example, a cube measuring Web hit volumes is dimensioned by the IP addresses of origin sites, the target URI, the referring sites, and hours in a day, as

define EXPvolume variable int <hour ip ref uri>

In designing the dimensions of the cube, we have to decide the finest level of granularity at which we want to do the analysis. For instance, we have chosen hours as the finest time granularity, even though the raw WLRs contain time data at an even finer granularity (seconds). The mapping between the fields of the WLR and the corresponding dimension values is referred to as *binning*.

Various cubes can be derived from the above basic cube as formulas. The ability to use formulas to define measures over a multi-dimensional space is a powerful feature of OLAP tools. Further, cubes can be computed from other cubes using *OLAP programming* in the scripting language provided by the OLAP engine.

Our infrastructure is illustrated in figure 1. WLRs may be kept in log files, or stored in the warehouse together with other reference data. In the latter case, WLRs are fed to the data warehouse periodically or continuously, and retired to archive after use, under data staging control.

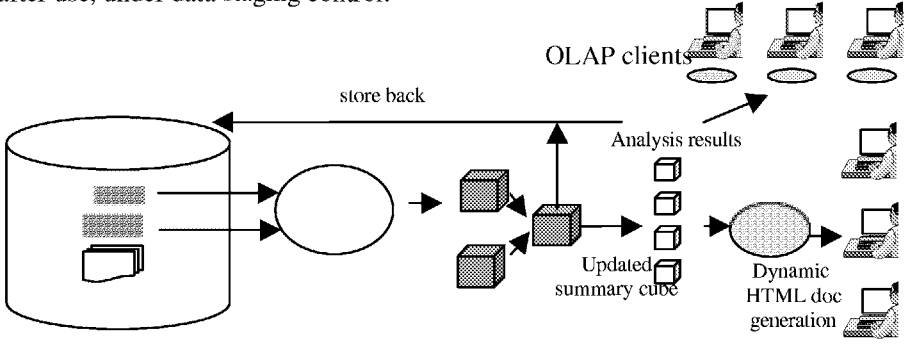


Fig. 1. OLAP-based Web access analysis automation

The infrastructure supports the computation of summary cubes and multidimensional and multi-level patterns and rules, based on both volume and probability distributions. The resulting summary cubes are stored back in the data warehouse, and reloaded into the OLAP server for incremental update. The entire operation chain from loading WLRs to computing summaries, patterns and rules, and posting the results on the Web is automated using OLAP programming.

3 Scaling Up Web Access Analysis

Scalability is the major challenge to any Web access analysis system. As we described in Section 1, one significant performance problem is the prohibitive cost of the traditional rollup operation for very large, sparse cubes. Application-level solutions to this problem are rarely provided. We have applied several 'non-traditional' OLAP operations to deal with this problem, which allow us to speed up Web log analysis by 3 orders of magnitude. Before explaining these solutions, let us first examine the typical cube rollup operation.

3.1 Typical Cube Rollup Operation

Elements of a dimension may form a hierarchy. A hierarchical dimension D contains elements at different levels of abstraction. Associated with D there are a dimension DL describing the levels of D, a relation DL_D mapping each value of D to the appropriate level, and a relation D_D mapping each value of D to its parent value (the value at the immediate upper level). To rollup cube C along dimension D, the measure value at a higher level is the total of the measure values at the corresponding lower levels. A cube may be rolled up along multiple dimensions.

In the application described in this paper, we consider *origin, subject, refsite* as high-level dimensions of *ip, uri, ref* respectively. In Oracle Express, the mappings between them can be defined by relations *origin_ip, subject_uri* and *refsite_ref.* Below are some mapping examples.

- *ip :63.211.140.164 → origin : CA*
- *uri:www.exp.com/TODAY/topstory.html→subject:www.exp.com/TODAY/*
- *ref: www.yahoo.com/entertaintment/book/book-store → refsite: www.yahoo.com/*

In the traditional OLAP approach, one defines a cube with multiple hierarchical dimensions, where each dimension has elements at more than one level. For example, one can define a dimension *from-site* with elements at *ip* level and *origin* level, drawn from dimensions *ip* and *origin* respectively; a *to-site* dimension with elements at *uri* level and *subject* level; a *via-site* dimension with elements at *ref* level and *refsite* level. The mappings between elements at different levels are based on the relations defined above. Then, a cube recording the volume of hits may be defined as

volume <from-site, to-site, via-site, hour>

When this cube is rolled up over all dimensions, it contains all the sub-totals of the original cells, for multiple dimensions and at multiple levels; this is referred to as

embedded-total. When the original cube has multiple large-sized dimensions, a large number of additional cells are needed to hold the embedded-total. In the above example, these sub-totals occupy approximately 50 trillion cells in the rolled up cube, many of them nulls, out of a total of 267 trillion cells. While the OLAP engine is designed to compress sparse cubes for storage, the cells containing nulls must be checked in some way during the rollup operation. Consequently, handling and rolling up such a cube *as a whole* is impractical.

3.2 Scalability Enhancements with Diagonal Aggregation

Our solution to the above problem is simply not to manipulate such a cube with large-sized dimensions *as a whole*. We define another, relatively smaller, cube to hold aggregated values with the following basic requirements: it provides high-level abstraction; and it maintains the relationships between dimension elements at different hierarchical levels to allow drill-down.

As shown in Figure 2, we represent Web access volumes at basic and aggregate levels by the following separate cubes.

- Basic Volume Cube (BVC), which takes into account all individual WLRs. This is the cube we defined earlier

  EXPvolume variable int <hour sparse <ip ref uri>>

- High-Diagonal Cube (HDC), which represents the summary information with respect to the parent dimensions of *ip, uri*, and *ref*, i.e. *origin, subject, refsite* respectively, as well as the hour dimension. N:1 mappings from *ip* to *origin*, from *uri* to *subject*, and from *ref* to *refsite*, are provided. The HDC in our example is defined as

  EXPvolume.high variable int <hour sparse <origin refsite subject>>

The HDC, *Expvolume.high*, is a summarization of the corresponding BVC, *EXPvolume*, aggregated *over all dimensions*. In this way *Expvolume.high* contains much fewer cells than *Expvolume*, and hence is easy to manipulate with reasonable performance.

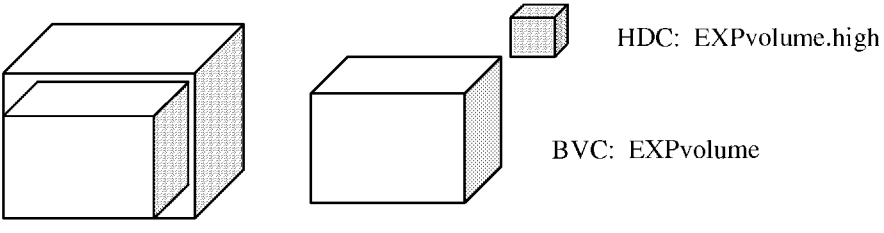


Fig. 2. Conventional cube rollup (left) vs. diagonal aggregation without rollup (right)

Note that *Expvolume.high* does not contain the partial aggregates of *Expvolume*, namely, the aggregates along one or more, but not all, dimensions. These aggregates can be selectively generated on demand as query results. For example, to drill down a *Expvolume.high* cell with `www.yahoo.com` as *refsite*, relation *refsite_ref* can be

used to relate it to a set of lower-level elements of dimension *ref*, such as `www.yahoo.com/entertaintment/book/book-store`, that underlie a sub-cube of *EXPvolume* (Figure 3). Since such query operations dealing with sub-cubes, they are relatively inexpensive.

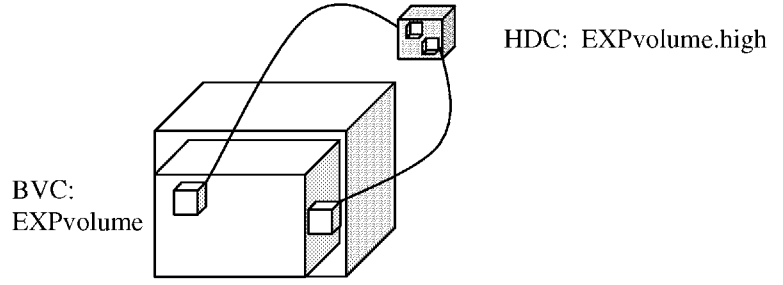


Fig. 3. Drill down from HDC to BVC based on query

3.3 Scalability Enhancements with Direct Binning rather than Rolling Up

Since *EXPvolume* is a sparse cube with large-sized dimensions, to generate the derived cube *EXPvolume.high* from it is rather expensive. Conversely, the WLRs, either stored in files or database tables, are not sparse. Very often, for each batch load, the number of WLRs is much less than the number of cells of the *EXPvolume* cube. For our application, we had millions of *WLRs*, but the *EXPvolume* cube had billions of cells. In this case, populating and updating a *EXPvolume.high directly from log files*, can reduce both memory load and computation load, compared with deriving it from *EXPvolume*. We call this mechanism *direct binning*.

Consider a simple case (Figure 4), where we have a volume cube with k dimensions $D_1,\ldots,D_k$, and each dimension is extended to include a single high-level element 'top'. To populate a summary cube containing the total as well as all the subtotals wrt each dimension element, each WLR contributes to $\sum_{i=0}^{k} C_k^i$ cells, where only one cell is for the base data, all others are for the above total and subtotals. In our example, the *EXPvolume.high* cube has 4 dimensions, therefore each WRL is used to update $C_4^0 + C_4^1 + C_4^2 + C_k^3 + C_k^4 = 16$ cells during direct binning. For sparse cubes with large dimensions, where the ratio between the numbers of high-level cells and input records falls within a certain range, directly populating high-level cells outperforms rollup.

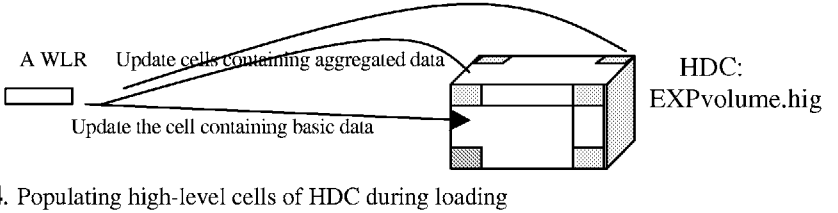


Fig. 4. Populating high-level cells of HDC during loading

3.4 Scalability Enhancement with High-profile Cubes

Limiting dimension elements to those that underlie cells with large counts, is another way to achieve data reduction. A cube defined on the limited dimensions contains fewer cells, thus is easy to manipulate. Using this approach, some cells containing small or zero counts will be dropped, but these are insignificant for most applications.

In our application, we introduced a high-profile cube (HPC) that is a sub-cube of HDC by taking into account only the the *high-profile elements* of dimensions *origin, subject, refsite*, i.e., those elements that correspond to Web access hit rates above a given threshold. The HPC in our example is defined as

EXPvolume.top variable int <hour sparse <toporigin toprefsite topsubject>>

The high-profile elements of a dimension are identified in the following way. Given a volume cube $C[D_1, \ldots, D_n]$ that measures hit counts, a dimension $D_i \in \{ D_1, \ldots, D_n \}$, a filter ratio $0 \le k \le 1$ wrt to the average count over the element of D_i, the threshold t is defined by the ratio of average counts per element of dimension D_i, *as* $total(C)/(size(D_i) * k)$, where $total(C)$ is the total counts of hits, and $size(D_i)$ is the number of elements in D_i. Those elements of D_i with subtotal counts over the threshold are considered "high-profile" ones. For example, for cube *EXPvolume.high*, the total counts can be calculated by the following.

EXPtotal = total(EXPvolume.high)

Dimensioned totals can be calculated by the following.

EXPbyorigin = total(EXPvolume.high, origin) //dimensioned by origin
EXPbysubject = total(EXPvolume.high, subject) //dimensioned by subject
EXPbyrefsite = total(EXPvolume.high, refsite) //dimensioned by refsite

Then, for example, the threshold for dimension *origin* is determined by

*threshold.origin = EXPtotal/size(origin) * k*

The high-profile elements of *origin* are extracted by

limit origin to EXPbyOrigin > threshold.origin

and then loaded to a separate dimension *toporigin*.

The elements of dimension *toporigin, topsubject* and *toprefsite* are subsets of those of *origin, subject* and *refsite* respectively. Therefore, total hits and their probability distributions must be calculated over *EXPvolume.high* for accuracy.

3.5 Overall Performance Comparison

In summary, we represent Web access volumes by separate volume cubes, a BVC (*EXPvolume*), an HDC (*EXPvolume.high*), and an HPC (*EXPvolume.top*). The table in Figure 5 shows how data can be reduced with this approach. It is easy to see from this table that without such data reduction, carrying out Web access analysis with OLAP is impractical. Figure 6 shows that the proposed approach outperforms the conventional approach dramatically. The comparison illustrates the practical value of our approach for handling the given class of application. With our approach, low-level details (measured by BVC) and high-level summaries (measured by HDC and

HPC), as well as links between them (measured by the relations between corresponding dimension elements), are made available. Information not directly covered by these cubes may be computed by queries involving relatively inexpensive sub-cube manipulations.

Dimension Sizes					
Ip	48,128	origin	90	toporigin	35
Uri	18,085	subject	229	topsubject	32
Ref	10,432	refsite	2,167	toprefsite	25
Hour	24	hour	24	hour	24
Cube Sizes					
EXPvolume	217,919 billion	EXPvolume.high	1.6 billion	EXPvolume.top	0.000672 billion

Fig. 5. Scalability improvement by data reduction

Conventional approach	
Loading cube EXPvolume	1 hour
Rollup Expvolume (by estimation)	10,000 hours
Total estimated time	10,000 hours
Proposed approach	
Loading Expvolume	1 hour
Direct binning Expvolume.high	1.2 hour
Generating EXPvolume.top	0.3 hour
Total time	2.5 hour

Fig. 6. A comparison of performance

4 Web Usage Analysis

We now describe the typical Web usage analysis functions supported by our system.

4.1 Multilevel and Multidimensional Analysis

As discussed above, cubes representing multidimensional Web access volumes are generated at three levels: the basic level, the summary level, and the top level. Various Web access patterns can be derived from these cubes. They may be used to represent the access behavior of a collection of users or a single user; they may be based on volumes or probability distributions; and they may be materialized (defined as variables) or not (defined as formulas). Below are some examples.

Multidimensional Patterns. A cube representing the access volumes by hour for the most popular subjects and top referring sites from users in France, can be defined by the following formula (view), derived from cube *Expvolume.top* by

define VolFromFrance.top formula int <hour, toprefsite, topsubject>
EQ EXPvolume.top(toporigin France)

Multilevel Patterns. Using the relations *origin_ip*, *refsite_ref*, and *subject_uri*, one can "drill down" from a specific cell in cube *EXPvolume.high*,

EXPvolume.high(hour '12,' origin France,' refsite 'www.yahoo.com/,'
subject 'www.exp.com/EXP/TODAY')

to identify a sub-cube of *EXPvolume*, through the following operations.

limit ip to origin_ip France'
limit ref to refsite_ref 'www.yahoo.com'
limit uri to subject_uri EXP/TODAY'
limit hour to '12'
report EXPvolume

Probability Distribution based Patterns. Cubes representing probability distribution based patterns are derived from volume-based pattern cubes. They provide fine-grained representation of dynamic behavior. Given cube *EXPvolume.high*, for example, the volume cube dimensioned by hour and subject is defined by

define VolByHourBySubject formula int <hour, subject>
EQ total(EXPvolume.high, hour, subject)

The cube representing probability distributions of the above information over all hits is expressed as

define VolByHourBySubject.dist1 formula decimal <hour, subject>
EQ total(EXPvolume.high, hour, subject) / total(EXPvolume.high)

Further, conditional probability distributions over the hits *per subject* is expressed as

define VolByHourBySubject.dist2 formula decimal <hour, subject>
EQ total(EXPvolume.high, hour, subject) / total(EXPvolume.high, subject)

In the actual implementation, some of the above cubes are materialized for computation efficiency. However, for consistency, it is only necessary to store volume cubes persistently in the data-warehouse. Derived patterns, either materialized or not, can be generated at analysis time.

4.2 Multi-level and Multidimensional Feature Ranking

Feature ranking, such as the top 10 Web sites being accessed, is important for such applications as targeted advertising. Using OLAP can enable ranking of Web access along multiple dimensions and at multiple levels. For example, given a Web site, e.g. `www.connect.com`, one may be interested in ranking the hits to it by companies, areas and time.

Given a volume cube, ranking on a feature (dimension) is represented by a pair of cubes, one for a ranked list of elements of that dimension, and the other for the cor-

responding volume or probability measures from which the ranking was computed. Consider the cube, *Expvolume.top* dimensioned by *hour, toporigin, toprefsite, top-subject*. The ranking of top N subjects dimensioned by origin sites, referring sites, and hour, is represented by the following pair of cubes

define subject_tp.list variable text <order hour toprefsite toporigin>
define subject_tp.perc variable dec <order hour toprefsite toporigin>

where "order" is a dimension containing ranks *1, 2, ... , N.*

In general, the multidimensional ranking information for a feature X is kept in a pair of ranking cubes dimensioned by $O, A_1, A_2 \ldots A_n$, say R_x $[O, A_1, A_2 \ldots A_n]$ and $R'_x[O, A_1, A_2 \ldots A_n]$. The ranked elements of X are kept as cell values of R_x, and the corresponding measure values (volume or percentage) are kept as cell values of R'_x. O is the dimension for ordered numbers *1,2, ... , N.* Typically these two cubes are computed from a measure cube with X and other dimensions related to $A_1, A_2 \ldots A_n$, denoted by C [X, $A'_1, A'_2 \ldots A'_m$]. The general algorithm is shown below.

In nested loops, focus on each subcube of $R_x[O, A_1, A_2 \ldots A_n]$ on dimensions $A_1, A_2 \ldots A_n$, say $R_x(A_1=a_1, A_2=a_2, \ldots A_n=a_n)$, that is dimensioned by O, denoted $R_{xs}[O]$. Map $R_{xs}[O]$ to a subcube of C, say, C_s, that is dimensioned by all dimensions except X. Generate a sorted list of X elements based on the measure values of C_s, and assign them to $R_{xs}[O]$. The corresponding measures (in volume or percentage) are assigned to the counterpart sub-cube of R'_x.

4.3 Correlation Analysis

An important aspect of Web access analysis is to understand the correlation between different factors such as between origin sites and subjects. Such correlations can be represented as association rules. We have described in [10] an approach to use cube operations to mine association rules, and we proposed several extensions to conventional association rules, including *scoped, multilevel, multidimensional rules.* We show how to apply these rules to web access analysis, and we describe further extensions to rules with *flexible bases* and *time-variant* rules.

4.3.1 Multilevel Association Rules with Flexible Base and Dimensions

Association rules provide a quantitative measurement of the correlation between facts [6]. For example, if 50% of the origin sites for accesses to pages belonging to some specific subject are via referring site www.yahoo.com, and only 10% of all these origin sites use Yahoo as a referring site, we say that the association rule has confidence 50% and support 10%. Given minimum support and confidence thresholds, a rule is considered *strong* if it satisfies these thresholds.

An association rule has an underlying *base* B that defines the population over which the rule is defined. For example, the correlation between subjects (i.e. target sites) and referring sites can be based on accesses, as

$x \in WLRs$: *contain_subject(x, S)* $\Rightarrow$ *contain_refsite(x, R),*

or based on origin site, as

$x \in origins$: *access_subject(x, S)* $\Rightarrow$ *via_refsite(x, R),*

regardless of whether the navigation occurs in the same session or not. In this example, the association rule uses binary predicates with the first place denoting a base element and the second place denoting an item.

In [6], we showed how to represent *multidimensional* and *multilevel* association rules using cubes. For example, the cube

C_v [*time, origin, refsite, subject*]

contains sufficient information for deriving association rules between referring sites and subjects (target sites). For example, we can define rules dimensioned by, and at different levels of *time* and *origin_area*, such as

[x∈origins: access_subject(x, S) ⇒ via_refsite(x, R)] | time = 'Jan99,' origin_area = CA'
[x∈origins: access_subject(x, S) ⇒ via_refsite(x, R)] | time = 'Year99,' origin_area = USA'

The above data cube also contains sufficient information for deriving rules that express the correlation between subjects, e.g.

[x∈origins: access_subject(x, A) ⇒ access_subject(x, B)] | time = '01Oct99,' origin_area = UK'

In [6], we gave an algorithm that, starting from a given volume cube such as C_v *(time, origin, refsite, subject)*, first computes a base cube C_b *(refsite, origin_area)*, a population cube C_p *(subject, refsite, origin_area)*, and an association cube C_a *(subject, subject2, refsite, time, origin_area)*; and then uses these cubes to derive support and confidence cubes. Note that the association cube includes a new dimension *subject2*, which has the same elements as *subject*, and its measure is the count of base elements corresponding to each combination of *subject* and *subject2*. We omit the details from this paper.

4.3.2 Time-variant Association Rules

In the above association rules, only the elements of the time dimension are considered. In reality, rules with respect to time-variant predicates may be more interesting, such as a rule that relates accesses (based on origin sites) to subjects A and B within the same day,

[x∈origins: access_subject(x, A) ⇒ access_subject(x, B)] | time = 'sameday',...

This rule concerns a predicate over the time dimension, which we model as a special dimension called *time-slot*.

The volume cube for computing association rules dimensioned by generic time-slots (e.g. same-day, same-week,...) is the same as defined above. The association, support, and confidence cubes are dimensioned by *time-slot*. Note that there is no need to dimension the population cube and base cube by *time-slot* since they are the same wrt all time-slot elements. The definitions of these cubes are shown below.

- *association cube:* C_a *(subject, subject2, refsite, time-slot, origin_area)*
- *population cube:* C_p *(subject, refsite, origin_area)*
- *base cube:* C_b *(refsite, origin_area)*
- *confidence cube:* C_f *(subject, subject2, refsite, time-slot, origin_area)*

- *support cube:* C_s *(subject, subject2, refsite, time-slot, origin_area)*

The computation of a rule dimensioned by generic time-slots differs from the computation of a rule dimensioned by time instants in the following aspects.

- Time bins: for rules dimensioned by generic time-slots, the time bins are not particular time elements such as hours or days, but rather time predicates. For instance, from an origin site, the accesses to subjects A and B in any week are mapped to the time slot element "same-week".
- Duplicate elimination: the base elements of rules, e.g. origin sites, are not repeatedly counted for a generic time-slot. For example, an origin site from which subjects A and B are accessed multiple times within the same day, or within the same day on multiple days, only contributes one count to same-day access.
- The handling of population and support cubes is also different, since they have no time–related dimensions as described above.

Accordingly, the algorithm for mining association rules dimensioned by generic time-slots includes the following additional or different steps from the algorithm described in [6].

- For each generic time slot (e.g., same-day), limit time instance accordingly (limit time to all days, excluding weeks, etc).
- The population cube C_P is instantiated with the dimensioned total counts of origin sites in each *origin_area*, wrt *subject, refsite, origin* and based on the antecedent condition C_v *(subject A)* > *0*. The base cube C_b is instantiated with the dimensioned total counts of origin sites wrt *refsite* and *origin_area.*
- In calculating the association cube C_a wrt each pair of *subject*, *subject2* , instead of counting the total origin sites that satisfy the association condition C_v *(subject A)* > *0* and C_v *(subject2 B)* > *0*, for each origin site (in a loop), check whether it satisfies that condition in any time instance belong to that time slot (e.g. any day), and if it does, count the origin site once only.
- Under the new definitions of these cubes, confidence cube and support cube are still computed by the cell-wise operations $C_f = C_a / C_p$ and $C_s = C_a / C_b$

5 Conclusions

Collecting and mining web log records (WLRs) from e-commerce web sites has become increasingly important for targeted marketing, promotions, and traffic analysis. We recognize that Web log mining solutions must scale to meet the requirements of huge data volumes and data flow rates encountered in these applications. We have identified several important scalability issues, and have developed an OLAP and data warehousing-based architecture that addresses these issues. We have applied several non-traditional approaches, allowing us to speed up WLR analysis by 3 orders of magnitude.

In addition to the typical OLAP operations, our framework also supports data mining operations such as multilevel and multidimensional association rules. We extended the class of association rules to include rules with flexible bases and time-variant rules. Finally, it is important in many applications to collect and analyze

Web logs continuously, rather than in batches. To this end, we have automated the whole operation chain, including data filtering, loading, incremental summarization and analysis.

Unlike most OLAP-based applications that treat OLAP servers purely as front-end analysis tools, we use the OLAP server as a computation engine, and support information staging between the data warehouse and the multi-dimensional database managed by the OLAP server. Further support for scalability can be provided through parallelism and distributed OLAP [6].

The infrastructure described in this paper has been fully implemented at HP Labs. Our results have validated the scalability and maintainability of this infrastructure.

References

1. Sameet Agarwal, Rakesh Agrawal, Prasad Deshpande, Ashish Gupta, Jeffrey F. Naughton, Raghu Ramakrishnan, Sunita Sarawagi, "On the Computation of Multidimensional Aggregates", 506-521, Proc. VLDB'96, 1996.
2. Torben Bach Pedersen, Christian S. Jensen, Curtis E. Dyreson, "Extending Practical Pre-Aggregation in On-Line Analytical Processing", 663-674, Proc. VLDB'99, 1999.
3. Stefano Ceri, Piero Fraternali, Stefano Paraboschi, "Data-Driven, One-To-One Web Site Generation for Data-Intensive Applications", 615-626, Proc. VLDB'99, 1999.
4. Surajit Chaudhuri and Umesh Dayal, "An Overview of Data Warehousing and OLAP Technology", SIGMOD Record Vol (26) No (1) , 1996.
5. Q. Chen, M. Hsu and U. Dayal, "A Data Warehouse/OLAP Framework for Scalable Telecommunication Tandem Traffic Analysis", Proc. of 16th International Conference on Data Engineering (ICDE-2000), 2000, USA.
6. Q. Chen, U. Dayal, M. Hsu, "A Distributed OLAP Infrastructure for E-Commerce", Proc. Fourth IFCIS Conference on Cooperative Information Systems (CoopIS'99), 1999, UK.
7. Daniela Florescu, Alon Y. Levy, Dan Suciu, Khaled Yagoub, "Optimization of Run-time Management of Data Intensive Web-sites", 627-638, Proc. VLDB'99, 1999.
8. Dimitrios Gunopulos, George Kollios, Vassilis Tsotras, Carlotta Domeniconi, "Approximating multi-dimensional aggregate range queries overreal attributes", Proc. ACM-SIGMOD'00, 2000.
9. J. Han, S. Chee, and J. Y. Chiang, "Issues for On-Line Analytical Mining of Data Warehouses", SIGMOD'98 Workshop on Research Issues on Data Mining and Knowledge Discovery (DMKD'98) , USA, 1998.
10. H. V. Jagadish, Laks V. S. Lakshmanan, Divesh Srivastava, What can Hierarchies do for Data Warehouses? 530-541, Proc. VLDB'99, 1999.
11. S. Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, Andrew Tomkins, "Extracting Large-Scale Knowledge Bases from the Web", 639-650, Proc. VLDB'99, 1999.
12. Net.Genesis `http://www.netgenesis.com`.
13. WebTrends, `http://www.webtrends.com`.

PROMISE: Predicting Query Behavior to Enable Predictive Caching Strategies for OLAP Systems[1]

Carsten Sapia

FORWISS (Bavarian Research Center for Knowledge-Based Systems)
Orleansstr. 34, D-81667 Munich, Germany
Email: sapia@forwiss.de

Abstract. This paper discusses the PROMISE (**Pr**edicting User Behavi**o**r in **M**ultidimensional **I**nformation **S**ystem **E**nvironments) approach, that deploys information about characteristic patterns in the user's multidimensional data access in order to improve caching algorithms of OLAP systems. The paper motivates this approach by presenting results of an analysis of the user behavior in a real-world OLAP environment. Further contributions of this paper are a model for characteristic OLAP query patterns based on Markov Models and a corresponding OLAP query prediction algorithm.

1 Introduction

Online Analytical Processing (OLAP) systems offer capabilities to interactively formulate queries by iteratively applying multidimensional operations (e.g. slicing, drilling). Due to the importance of interactive response times in these environments, caching techniques for OLAP systems have received the attention of researchers ([1], [4], [11]). Additionally, it has been recognized ([8]), that the workload of an OLAP application is characterized by the user´s explorative and navigational data analysis task. A session typically starts with a query (mostly a predefined report) and the user then successively manipulates the results applying multidimensional operations (each of which results in a new query against the database). As a consequence, the OLAP workload shows specific high-level patterns that stem from the structure of the analytical task the user is solving.

The idea of the *PROMISE* (**Pr**edicting User Behavi**o**r in **M**ultidimensional **I**nformation **S**ystem **E**nvironments) approach is to provide the OLAP cache manager with information about these high level access patterns in order to make predictions. An algorithm that (at a point during an OLAP session) efficiently computes a set of queries and the corresponding possibilities for these queries to be executed in the near future can enhance caching techniques in two ways:

Admission and eviction algorithms both need functions to estimate the benefits of a cached object for future queries ([11]). At this point, an accurate prediction of future queries and their probability can be used to enhance the computation of the future benefit of a cached object.

[1] An extended version of this paper can be found in [9]

Y. Kambayashi, M. Mohania, and A M. Tjoa (Eds.): DaWaK 2000, LNCS 1874, pp. 224–233, 2000.
© Springer-Verlag Berlin Heidelberg 2000

Additionally, *Prefetching strategies* can complement the demand fetching strategies currently deployed for OLAP caches. Making use of predictions the cache manager can speculatively fetch objects into the cache (during idle times) that are likely to be beneficial for future queries. This results in a reduction of the latency time perceived by the user when the prefetched data is actually accessed.

As the example scenario for the rest of this paper, let us consider a logistics planning system used to plan the supply of spare parts for different kinds of vehicles (automobiles, trucks etc.). A data warehouse contains data about which vehicles are deployed in which regions, information about repairs of vehicles (including which parts were exchanged, what type of failure occurred, the geographical location of the repair). Figure 1 depicts an extract of the multidimensional schema for such an application. A typical task of a logistic planer is to determine a distribution and stock-keeping strategy for a certain geographic region or repair location (e.g. how many parts are kept locally to satisfy anticipated demands). While this task is far from being trivial and cannot be easily automated, the analysis process follows a methodological structure of subtasks.

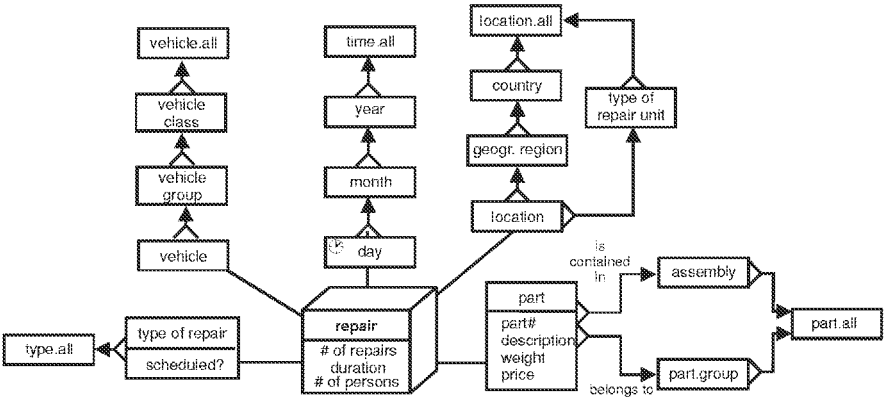


Fig. 1. The schema for the example scenario (depicted using the ME/R notation [10])

2 Related Work

To the best of our knowledge no work on predictive prefetching in OLAP environments has been done so far. Nevertheless, predictive prefetching techniques have been extensively researched in the areas of branch prediction for pipelined microprocessors, operating systems in order to do intelligent prefetching of files (e.g. [5]), distributed hypertext applications (e.g. WWW) for presending documents and proxy cache design (e.g. [2], [7]) and for OO databases (e.g. [3]) to prefetch objects.

Directly transferring these approaches to the OLAP domain, where the entity of prediction is a multidimensional query, shows two fundamental differences of OLAP systems compared to the other application areas: First, the number of prediction entities (queries) is considerably larger than in other domains (e.g. the number of files in a system). Second, different cached objects (i.e. queries) are not disjoint in the sense,

that results of a new query can be computed using parts of several cached objects. That means, a prediction is already useful if a query is being predicted from which the exact query can be derived. Another distinctive feature of PROMISE compared to all the presented approaches is the usage of patterns on a level of abstraction higher than the actual granularity of access (e.g Web documents, Pages, Objects, Files) – see section 5.

All of the current OLAP caching approaches ([1], [4], [11]) assume a demand fetching strategy, therefore, we view our work complementary to these caching algorithms.

3 Empirical Analysis of OLAP System Workloads

In order for predictive prefetching techniques to be successful, the workload has to be navigational and the consideration time between two accesses must be long enough to allow for prefetching the results. The real-world system that has been investigated regarding these criteria is an SAP BW system supporting the distribution logistics of the material management division in a large chemical company. The interaction behavior of 18 users was monitored over a two month period including 260 sessions containing 3150 queries.

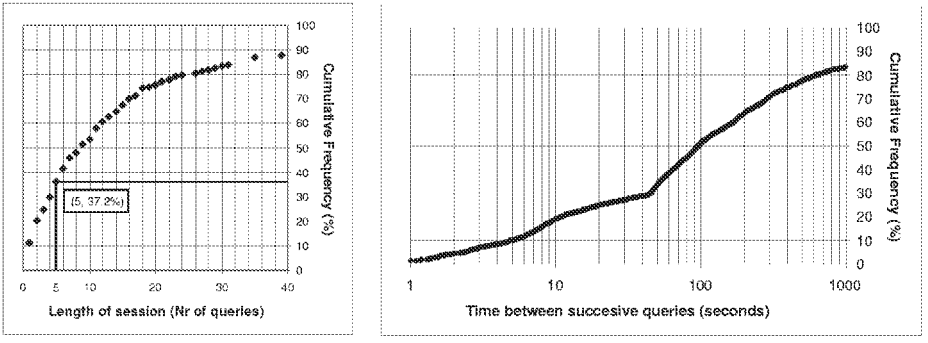


Fig. 2. Cumulative distributions of session length and consideration time

In OLAP systems, the *navigational nature* of the workload is guaranteed as long as the user interactively formulates his next request using the results of the previous request ([9]). We call such a sequence of navigational queries a session. The analysis showed that typical OLAP sessions have a considerable length and are thus suited for prediction approaches. The left hand side of Figure 2 shows the cumulative frequency distribution of the session length. It is obvious, that only 11% of the sessions consisted of executing a single query (simple reporting). On the other hand, some of the sessions contained more than 100 queries. If we assume that accurate prediction is possible for sessions with 5 or more consecutive queries, Figure 2 shows that 63,8% of the sessions fulfill this condition.

Additionally, the analysis showed that the *consideration time* between two queries is long enough for a significant percentage of queries. The right-hand side of Figure 2 plots the cumulative frequency distribution of the consideration time against a loga-

rithmic scale. If we assume that the prediction and execution of a query takes 10 seconds (The median for query execution times is 9,4 seconds), over 81% of the queries have a consideration time that is long enough for prefetching. These results are confirmed when directly comparing consideration time and execution time for individual queries: 7% of the queries in our scenario cannot benefit from prefetching because they start a session. For 8% of the queries, the execution time of the next query is larger than the consideration time of the user, forbidding a full prefetch (rapid queries). The remaining 85% of the queries analyzed, could potentially fully benefit from predictive prefetching.

4 Terminology and Basic Concepts for PROMISE

This section introduces Markov Models (e.g. [6]) as a formalism to model interactive behavior. It also motivates which steps are necessary to adapt Markov Models to the OLAP domain. Furthermore, it presents the formal model for MD schemata and presents the class of multidimensional queries considered for the PROMISE approach.

Discrete time Markov Models (DTMM) represent a system as a finite set of states $S=\{s_1,...,s_n\}$. At each discrete point *t* of the (logical) timescale exactly one of the states $s^t \in S$ is active. Between two points in time (t-1) and t, the system changes its active state from $s^{t-1} \in S$ to $s^t \in S$. The probability that a specific state s_i becomes the next active state is determined by a transition probability function p. The transition probability is only dependent on a fixed number of states *m* (called the order of the DTMM) that have been visited directly before the point *t* (i.e. the states s^{t-1}, s^{t-2}, s^{t-m}).

Definition (Order-m Markov Model): An order m Markov Model is a tuple (S, P) where S denotes a finite number of states and P: $S \times S^m \rightarrow [0;1]$ denotes the probability function for state transitions. ◆

Designing a basic prediction algorithm based on a DTMM is rather straightforward. During a session, the vector $v=(s^{t-m}, ...,s^{t-1})$ containing the last m states that have been visited has to be recorded. The transition function $P(s^t,v)$ can then be used to calculate the most probable states to be visited next. Normally each state transition corresponds to an interaction (e.g. the activation of a WWW link). When applying this to the OLAP domain, each state transition would be the execution of a multidimensional query as the result of applying an MD operation. This naive approach has severe technical and conceptual drawbacks: *Large State Space*: Even if we restrict the class of queries, the number of queries that can be executed against an MD schema is still orders of magnitude larger than e.g. the number of documents of a WWW site. A large number of states leads to a large set of potential successor states which in turn decreases the probability of the transitions. Therefore, this model is not only very costly to maintain but also not very predictive. *Higher Level Patterns cannot be exploited*: The fine grained approach only predicts mere repetitive patterns that consist of exactly the same queries. This is unlikely for OLAP applications where e.g. the same sequence of query templates are being executed with different parameters.

Section 6 will show how our approach tackles these problems. But first we have to give a definition of the multidimensional data model which we need later for the definition of a canonical query. We only describes the most important elements of the

MD data model which are necessary to understand the prediction algorithm. A more comprehensive version of this data model can be found in [8].

Definition (Dimension Schema): A *dimension schema d* is defined as tuple $d := (L_d, class)$ with L_d being the set of level names. Each level $l \in L_d$ has a finite domain *dom(l)* attached. Furthermore, every dimension schema contains a special level *<dimension_name>.all* which contains only the special member <all>. The relationship $class \subseteq L_d \times L_d$ defines the classification relationship between levels. $(l_1,l_2) \in class^*$ reads "l_1 can be classified according to l_2." ◆

Example: The 'location' dimension of our example (cf. Fig. 2) can be formalized as follows $d_{location}$:= ({ location, type_of_unit, geogr_region, country, location.all},
{(location, type_of_unit), (location, geogr_region), (geogr_region, country),
(country, location.all), (type_of_unit, location.all) }) ◆

The basic organizational element of an MD database is a multidimensional cube which contains the interesting measures. The schema of an n-dimensional cube is defined by combining n dimension schemata with a set of measures.

Definition (Multidimensional Cube Schema): An *n-dimensional cube schema* Ω is a tuple $\Omega = (D_\Omega, M_\Omega)$ where $D_\Omega = (d_1,..d_n)$ is a list of dimension schemata and M_Ω is the set of measures of the cube. ◆

For notational convenience, we use the following conventions: *dom(d)* denotes the domain of a dimension $d \in D$. All elements $x \in dom(d)$ are called *dimension members* of dimension *d*. Furthermore, Ψ denotes the set of all levels of a cube schema Ω. Thus,

$$dom(d) = \bigcup_{l \in L_d} dom(l) \qquad \text{and} \qquad \Psi = \bigcup_{d \in \Omega} L_d$$

The function *level(x)* returns the dimension level to which a dimension member belongs, i.e.: $level(x) = l :\Leftrightarrow x \in dom(l), l \in \Omega, x \in \bigcup_{d \in D} dom(d)$.

The schema of a multidimensional database determines the queries that can be executed by the user. Our approach relies on a canonical representation of an OLAP query. Such a query is defined by the user selecting a set of interesting measures first (called *result measures*). Then, for each dimension of the cube schema Ω, the user picks a *restriction element* (which may be the *all*-element representing no restriction). Implicitly, this defines a restriction on the base elements of the dimension. Figure 3 visualizes the selection process for a two dimensional data space. In dimension 1 (left), the restriction element is on level $L_{1,1}$ of the dimension hierarchy. The structure of the result is defined by giving a *result granularity* for each dimension. This implicitly defines the necessary aggregations that have to be performed to the base data stored in the cube. For example, in Figure 3, for dimension 1 (left) the result granularity is $L_{1,2}$. The result of the query is shown on the right hand side of Figure 3.

Definition (Canonical OLAP Query):
A *canonical OLAP query* q_Ω over an n-dimensional cube schema $\Omega = ((d_1,..,d_n), M_\Omega)$ is defined as a 3-tuple $q_\Omega := (M, R, G)$ where $M \subseteq M_\Omega$ is the *result measure set* and. $R = (r_1, r_2, \ldots, r_n)$ with $r_i \in dom(d_i), 1 \le i \le n$ is called the *restriction vector* of q_Ω.

The element r_i is called the *restriction element* for dimension d_i. The set $G = (g_1, g_2, \ldots, g_n)$ with $g_i \in \Psi$ is called the *result granularity vector* of q_Ω. In order for q_Ω to be valid, the restriction must not be finer than the result granularity, thus $(level(r_i), g_i) \in class^*_{d_i}$ $\forall 1 \le i \le n$. For a cube schema Ω we denote the set of all possible canonical queries as Θ_Ω ◆

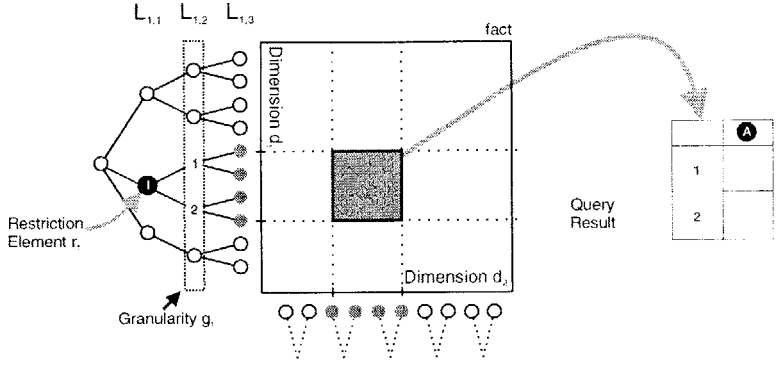


Fig. 3. Visualization of a canonical query on a two dimensional cube schema

Example: An example query against the schema Ω_{repair} might be "Give me the number of repairs for the year '1999' and the 'steering' assembly split up by the geographic regions of 'Germany'".

q_1:= ('#repairs', ('Germany', '1999', 'all', 'all', 'steering'),
(geogr. region, year, type.all, vehicle.all, assembly)) ◆

5 Abstracting to Patterns

This section presents different abstractions of canonical queries and argues why these are likely to produce more significant patterns for interactive data analysis applications. Mathematically speaking, the abstraction is defined by an equivalency relation $\alpha \subseteq \Theta_\Omega \times \Theta_\Omega$ on the set of all canonical queries Θ_Ω. (We denote the equivalency class of each query $q \in \Theta_\Omega$ as $\wp(q)$). For PROMISE we distinguish structural and value-based patterns.

Structural patterns mirror the fact that each stage of the analysis process has a set of characteristic views on the data. E.g. during the first phase of the analysis the logistics expert is looking for irregularities of failures according to different geographical regions. This means he compares the failure rate of different assemblies in the different regions. Thus, the current interest of the analyst can be derived from the structure of the query. First, it is important to note on which level of granularity the user formulates his queries (i.e. the granularity of the result G). A second hint to the users intention is his choice which dimensions are result dimensions (visualized e.g. as a table) and which dimensions are restricted to a single value (selection dimension). The rationale behind this assumption is that if the user is analyzing dependen-

cies between dimensions (e.g. geographic region and broken parts), the user will choose theses dimensions as result dimensions.

Definition (Selection Dimension):

For a canonical OLAP query q_Ω= (M, (r_1,...,r_n), (g_1,...,g_n)) against a cube schema Ω=((d_1,...,d_n), M_Ω), we define the set of *selection dimensions* σ_q as follows: $\sigma_q := \{d_i \mid level(r_i) = g_i\}$. A dimension which is not a selection dimension is called *result dimension.* ◆

Following the above description of structural patterns, we define an abstraction of a query (called Structural Query Prototype) that only considers the measures contained in a query, the granularity of the result and which of the dimensions are selection dimensions and which are result dimensions. This prototype abstracts from the actual values used to restrict the query.

Definition (Structural Query Prototype):

Let us assume a query q_Ω = (M, R, G) with Ω=((d_1,...,d_n), M_Ω) and its set of selection dimensions σ_q. The structural query prototype $\wp_q$ for this query is defined as follows:

$$\wp(q) := (M, G, \Sigma = (\Sigma_1, \ldots, \Sigma_n)) \text{ with } \Sigma_i = 1 \text{ if } d_i \in \sigma_q \text{ and } \Sigma_i = 0 \text{ if } d_i \notin \sigma_q$$

We denote the set of all structural query prototypes for an MD schema Ω as $\wp_\Omega$. ◆

Example: The query q_1 = "Give me a ranking of garages in Germany according to the number of repairs summed up to the geographic region for 1998" and the query q_2 = "Give me a ranking of all garages according to the number of repairs summed up to the geographic region for 2000"

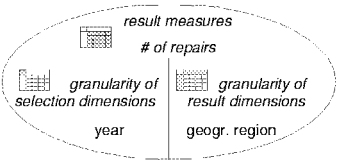


Fig. 4. Query Prototype

both have the same prototype: $\wp_{q1} = \wp_{q2}$= ({#repairs}, (geogr.region, year, vehicle.all, type.all, part.all), (0,1,1,1,1)}) ◆

Figure 4 shows a graphical representation of a structural query prototype for this example. With a view to greater clarity, we omit the 'all'-levels from the enumeration of selection levels.

Value-based patterns model the fact that the analyst often accesses values of a dimension in a certain order which may be problem specific. E.g. the analyst may start the analysis with his own region and then analyze the adjoining regions. In order to describe these value-based patterns between successive queries, we define a value-based equivalency relation. As these patterns are specific for a single dimension, the equivalency relation is defined dimension specific:

Definition (Value Based Abstraction):

Assuming two queries $q = (M, \{r_1, \ldots, r_n\}, G)$ and $\overline{q} = (M, \{\overline{r}_1, \ldots, \overline{r}_n\}, G)$, the value-based equivalency relation α_V^i is defined as: $\alpha_V\ (q, \overline{q}) :\Leftrightarrow r_i = \overline{r}_i$ ◆

6 The PROMISE Prediction Algorithm

The previous section presented different abstract views on the interaction sequences. The purpose of the abstraction was to suppress detail information about a query in order to produce more significant, higher level patterns. The problem for prediction is

that a single prediction model which is built only on the abstract view can only make abstract predictions. This section shows how to overcome this problem by using a set of prediction models and exploiting the similarity of two subsequent queries.

Summarizing our considerations presented so far, we base the prediction model on the following central assumptions:

1. The structural patterns are independent of the restriction values (i.e. the probability that the next query has a certain structural prototype, is not dependent on the current restriction elements).
2. Patterns for the restriction values in the different dimensions are independent of each other (i.e. the probability of selecting e.g. '1999' next is not dependent on the current restriction e.g. in the location dimension).
3. The probability which of the restriction elements change between two queries is dependent on the structural protoypes of the two queries.

Assumption (1) and (2) allow us to use (n+1) independent Markov Models (one *Structural Model* and one *Value-Based Model* for each dimension) for the prediction. Assumption (3) is a refinement of the separation to increase the accuracy of the prediction (see below).

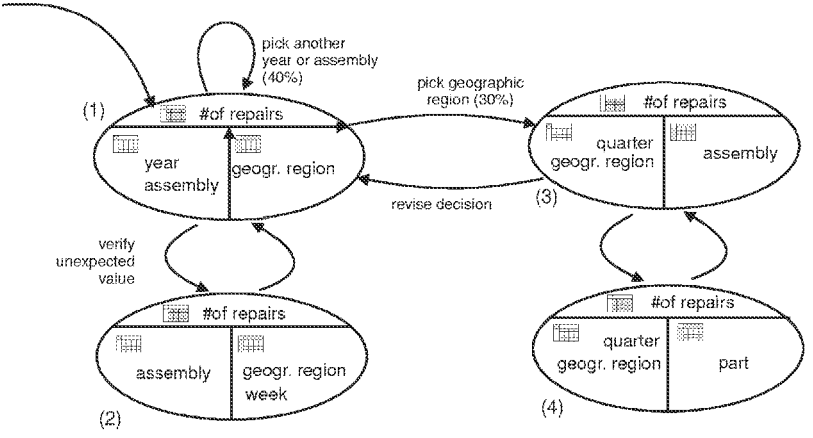


Fig. 5. A sample Structural Prediction Model (SPM)

The state space of the *Structural Prediction Model (SPM)* is defined by the set of structural prototypes of a model. A transition from state $\wp_1$ to $\wp_2$ is labeled with the probability that a query q_1 with prototype $\wp_1=\wp(q_1)$ is followed by a query q_2 with prototype $\wp_2=\wp(q_2)$. Figure 5 shows an extract of the structural prediction model for the example schema Ω_E. The *SPM* mirrors the macro structure of the analysis process. It can be used to predict the selection levels, the result granularity and the result measures of the next query. As it does not contain any information about values, it cannot predict the restriction elements.

For the prediction of restriction values, we introduce a *Value-Based Prediction Model (VPM)* for each dimension. This model contains information about typical navigation pathes in a dimension. The states are labeled with the dimension members and transitions from state a to state b represent a sequence of queries q,p where q was

restricted by *a* and *p* was restricted by *b*. Figure 6 (left) shows a sample *VPM* for the *'location'* dimension. As stated in assumption (1), the *VPM* is independent of the *SPM* and is used to predict restriction values for the next query.

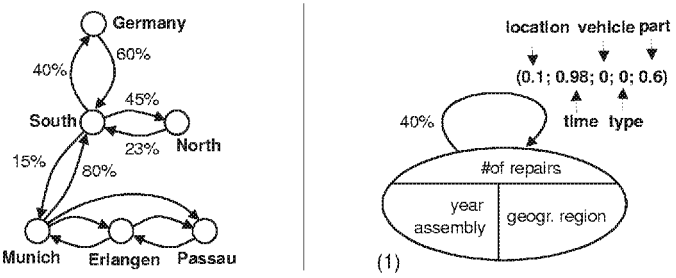


Fig. 6. Examples for a VPM (left) and a probabilistic change vector (right)

So far, we can already predict a complete next query using the combination of SPM (result measures, result granularity) and VPM (restriction elements). The accuracy of the prediction can be increased by taking into account assumption (3). It is based on the observation, that typically not all of the restriction values are changed between two subsequent queries. Which dimensions change restriction elements is often dependent on the current state of the analysis process (i.e. the structure of the last query). Therefore, we extend the SPM to additionally record the probability for the different restriction values to change (called *probabilistic change vector*) for every transition. For example, Figure 6 (right) shows the probabilistic change vector for a transition of the sample SPM. The probability that the restriction element for dimension 1 (location) changes between two subsequent queries with a structure corresponding to state (1) is 10%. This information is used to further restrict the set of candidate restriction values for the dimension in the following way: If the probabilistic change vector indicates a probability that is below a certain threshold (e.g. 15%), it is assumed that the restriction value for this dimension does not change. The following formal definition summarizes the above considerations:

Definition (Prediction Profile): A Prediction Profile P_Ω for an n-dimensional cube schema Ω is a 3-tuple (SPM, ϖ, VPM) with *SPM* being a Markov Model containing the structural prototypes of the cube schema as states. $SPM = (\{\wp(q) \mid q \in \Theta_\Omega\}), P_S)$ is called the *structural prediction model*. The function $\varpi: \wp_\Omega \times \wp_\Omega \rightarrow [0,1]^n$ assigns a probabilistic change vector to every transition of the structural prediction model *SPM*. $VPM = (vpm_1,...,vpm_n)$ is a set of n Markov Models. Each $vpm_i = (\{r \mid r \in dom(d_i)\}, P_{Vi})$; $i \in [1;n]$ is a Markov Model based on the values of dimension d_i and is called the *value prediction model* for dimension d_i. ◆

Prediction Profiles can either be built by domain experts during a conceptual modeling process ([8]) or can be induced from query log files that contain the queries that were executed in the past. The prediction algorithm uses such a prediction profile in the following way: We assume that the system keeps track of the last query of the user's session $q^{t-1}=(M^{t-1}, R^{t-1}, G^{t-1})$. The algorithm computes a set of candidates for the query $q^{t-1}=(M^t, R^t, G^t)$.

First, we use the *SPM* together with q^{t-1} to predict the granularity vector G^t and the result measures M^t of the next query. In order to do sensible prefetching, we additionally need to predict the selection vectors R^t. The prediction of R^t consists of two phases. For the selection dimensions of q^t, r_i^t can be derived from the probabilistic change vector $\omega(\wp(q^{t-1}), \wp(q^t))$ in the following way: if the probability for a value change in the i^{th} dimension (ω_i)is below a certain threshold, s_i^t is assumed to be equal to s_i^{t-1}. If ω_i is higher, the prediction model VPM_i is being used to predict the next selection value. If d_i is predicted to be a selection dimension in q^t, we additionally know, which granularity level, the next restriction element will belong to as the level of selection is equal to the granularity of the result g_i^t. This additional information is used to provide a better prediction as the potential successor states in VPM_i are restricted to values of $dom(g_i^t)$.

Example: May query q^{t-1} be the query shown in the example in section 4.3 (this query corresponds to state (1) in Figure 5). The prototype predicted by the SPM (with a confidence of 40%) is $\wp(q^t)$=({#repairs}, (geogr_region, year, vehicle.all, type.all, assembly), (0,1,1,1,1)) which also corresponds to state (1) of the SPM. If we assume a value change threshold of 0.15, the restriction elements of dimensions 1 (location), 3 (vehicle) and 4 (type) are assumed not to change (cf. Figure 6) while the VPM's are consulted for the selection values s_2^t (time) and s_5^t (part). Being selection dimensions, in both cases it is known that s_2^t belongs to the level year and s_5^t belongs to assembly.

References

[1] J. Albrecht, A. Bauer, O. Deyerling, H. Günzel, W. Hümmer, W. Lehner, L. Schlesinger: *Management of Multidimensional Aggregates for Efficient Online Analytical Processing*. Proc. of the IDEAS, 1999.

[2] A. Bestavros: *Speculative Data Dissemination and Service*, in Proc. of the 19th International Conference on Data Engineering (ICDE), 1996.

[3] K.M. Curewitz, P. Krishnan, J.S. Vitter: *Practical Prefetching via Data Compression*, in Proc. of the SIGMOD '93, ACM Press, 1993

[4] P. Deshpande, K. Ramasamy, A. Shukla, J.F. Naughton: *Caching Multidimensional Queries Using Chunks*. In Proc. ACM SIGMOD Conference, pp. 259-270, 1998.

[5] J. Griffioen, R. Appleton: *The Design, Implementation, and Evaluation of a Predictive Caching File System*, Technical Report No. CS-264-96, University of Kentucky, 1996.

[6] R.A. Howard: *Dynamic Programming and Markov Processes*, John Wiley, 1960.

[7] V. N. Padmanabhan, J. C. Mogul: *Using Predictive Prefetching to Improve World Wide Web Latency*. In Proc. of ACM SIGComm '96, pp.26-36, 1996.

[8] C. Sapia *On Modeling and Predicting User Behavior in OLAP Systems*, Proceedings of the DMDW99 Workshop, CAiSE Conference, 1999.

[9] C. Sapia *PROMISE – Modeling and Predicting User Query Behavior in Online Analytical Processing Environments*, FORWISS Technical Report FR-2000-001, June 2000. http://www.forwiss.de/public/reports.html.

[10] C. Sapia, M. Blaschka, G. Höfling, B. Dinter, *Extending the E/R Model for the Multidimensional Paradigm*, in "Advances in Database Technologies", LNCS Vol. 1552, 1999.

[11] J. Shim, P. Scheuermann, R. Vingralek, *Dynamic Caching of Query Results for Decision Support Systems*, in Proc. of the SSDBM Conference, 1999.

Supporting Online Queries in ROLAP *

Daniel Barbará Xintao Wu

George Mason University
Information and Software Engineering Department
Fairfax, VA 22303
{dbarbara,xwu}@gmu.edu

Abstract. Data warehouses are becoming a powerful tool to analyze enterprise data. A critical demand imposed by the users of data warehouses is that the time to get an answer (latency) after posing a query is to be as short as possible. It is arguable that a quick, albeit approximate, answer that can be refined over time is much better than a perfect answer for which a user has to wait a long time. In this paper we addressed the issue of online support for data warehouse queries, meaning the ability to reduce the latency of the answer at the expense of having an approximate answer that can be refined as the user is looking at it. Previous work has address the online support by using sampling techniques. We argue that a better way is to preclassify the cells of the data cube into error bins and bring the target data for a query in "waves," i.e., by fetching the data in those bins one after the other. The cells are classified into bins by means of the usage of a data model (e.g., linear regression, log-linear models) that allows the system to obtain an approximate value for each of the data cube cells. The difference between the estimated value and the true value is the estimation error, and its magnitude determines to which bin the cell belongs. The estimated value given by the model serves to give a very quick, yet approximate answer, that will be refined online by bringing cells from the error bins. Experiments show that this technique is a good way to support online aggregation.

* This work has been supported by NSF grant IIS-9732113

Y. Kambayashi, M. Mohania, and A M. Tjoa (Eds.): DaWaK 2000, LNCS 1874, pp. 234–243, 2000.
© Springer-Verlag Berlin Heidelberg 2000

1 Introduction

A *data cube* is a popular organization for summary data [12]. A cube is simply a multidimensional structure that contains at each point an aggregate value, i.e., the result of applying an aggregate function to an underlying relation. For instance, a cube can summarize sales data for a corporation, with dimensions "time of sale," "location of sale" and "product type".

The underlying relation that serves as the base for the cube creation is commonly referred to as *fact table* in the *star schema* [16]. The fact table contains all the attributes that determine the dimension of the cube, plus an attribute on which the aggregations are to be performed. Along with this table, there are *dimension tables* whose keys are foreign keys of the fact table and which describe each one of the dimension attributes. The combination of fact table and dimension tables form what is called a *star schema*. In our example, the fact table would contain the individual sale tuples, stating the time of sale, location, product and amount. The last attribute (amount) is the one on which the aggregations are to be performed (to find out, for instance, the total sales amount for a particular location, during the month of January, 2000).

A data cube is composed by a series of aggregation levels that form a lattice. (see [12, 13]). The basic level, *core cuboid*, contains aggregations at the finer level (i.e., where each dimension of the cube is represented by a value). For instance, in our example the core cuboid would contain one cell for each combination of time, location and product type. The value contained in that cell would be the aggregate of each individual tuple in the fact table whose attributes time, location, and product type are precisely those of that cell.

Each tuple in the fact table contributes to one cell of the core cuboid. However, not all cells in the core cuboid have corresponding tuples in the fact table. There exist (usually many) cells whose values are null, simply because there is no tuple in the fact table that exhibits their combination of attributes. In our example, there will be combinations of time, location and product type for which no tuple exists, and therefore the corresponding cell value is null. Coarser aggregated cells (e.g., time, and location for **all** products), form other cuboids in the data cube.

Multidimensional data bases are a central concept in the field commonly known as On-Line Analytical Processing (OLAP). OLAP systems are typically implemented either directly on top of relational systems or as a combination of a relational system and a proprietary multi-dimensional database (MOLAP) [11]. In both approaches the underlying database is stored in a relational system. In a fully relational implementation (ROLAP), data cube queries are transparently converted into queries on the underlying relational database. Clever indexing schemes and careful query optimization are used to improve the performance of these queries. In the MOLAP approach, the cube data is extracted from the database, converted into multidimensional arrays, and clustered so that common cube queries require minimal I/O.

Queries (mostly aggregations) in OLAP are performed in a mode that mostly resembles a batch operation. A query is posed, the system processes it (usually

for a long period of time) and the answer is then returned to the user. This is particularly unfortunate in the OLAP environment, since OLAP is intended to serve as a decision support tool. The analyst is intent in finding facts about the organization, and more often than not, the results of the query drive the next query the analyst wants to ask. Waiting for a long time for an answer breaks the train of thought and is likely to render the system much less appealing for users. The analyst would be better served by a quick, yet approximate answer that gets refined as she is thinking about the significance of the data.

To that purpose, Hellerstein, Haas and Wang [15] have proposed and implemented a technique that allows users to observe the progress of the query and obtain a quick, approximate answer that gets refined over time. At any point in time, an error estimate is offered to the user. The user controls an interface that permits stopping the processing of the query at any point where she is satisfied with the answer. The authors do this by performing a running aggregate, retrieving records in random order and estimating the proximity of the aggregate to the final result as a running confidence interval. A follow-up paper [14] aims to optimize this technique by using *ripple joins*, designed to minimize the time until an acceptable precise estimate of the query result is available.

There are, however, several problems with the approach taken by [15], as pointed out in [1]. The use of sampling to estimate the output of a join, can produce a poor quality approximation, due to the non-uniform result sample (the join of two uniform samples is **not** a uniform random sample of the output of the join), and the size of the join result (the join of two random samples has very few tuples, leading to inaccurate answers and poor confidence bounds). The authors of [1] propose to remedy these problems by using precomputed samples of a small set of distinguished joins that serve as a basis to answer queries. However, it is not clear how such a solution can be used to implement a system that refines the answer as time goes by.

In this paper, we present an alternative method for supporting on-line aggregations in ROLAP. Our technique relies on the characterization of chunks of the data cube via statistical (or other type of) models. Partitioning the cube and fitting a model to each of the resulting chunks gives a way of estimating the values of the cells. Of course, these estimations are never perfect. However, some cells are better fitted by the model than others. Our technique classifies the tuples in the fact table in *error bins*, according to how far the estimates of their corresponding cells in the cube are from the real values of the tuples. Having done this, queries are processing first by using the models to estimate the values of the target cells and retrieving one after another, sets of target tuples in each error bin, beginning with the bin that corresponds to the highest error level, and following the bins in the descending error value. By doing so, the system can offer a quick, albeit approximate answer based in the estimates, which can be refined over time as tuples from the different error bins are processed. Since the errors in the bins have been pre-computed in advance, there is no need to estimate the confidence level.

This paper is organized as follows. Section 2 describes our technique. Section

3 presents an architecture of the system. Section 4 shows the results of experimental evaluation. Finally Section 5 offers conclusions and avenues for future work.

2 Our Technique

The basis for our technique is the concept of the *Quasi-Cube*, formulated previously by us in [6, 7, 9] as a way to achieve cube compression. Quasi-Cubes are the only one cube compression technique that allows the designer to establish guarantees for the errors incurred in the answers in such a way that the errors are kept below a predefined threshold, *regardless of the distribution of the data.* All the other techniques [6, 7, 9, 17, 19] can guarantee error levels, but the errors vary with the underlying data distributions and cannot be fixed by the designer.

In the Quasi-Cube, we model regions of the core cuboid and employ these models to estimate the values of the individual cells. The reason to focus on the core cuboid is simple: the error guarantees for queries to the core cuboid hold for any other cuboid in the lattice. (In practice, the errors incurred when aggregating estimated cells of the core cuboid decrease dramatically because they tend to cancel each other.) We achieve fixed levels of error by retaining cells of the cube whose values, when recreated by the decompression algorithms differ from the real values in more than the tolerated threshold.

In this paper, however, we use the Quasi-Cube idea for a different purpose. Once the modeling of the regions is finished, we classify and place each tuple in the fact table in an *error bin.* An error bin is a repository of tuples from the fact table that contribute to cells in the core cuboid whose estimated values differ from their real values by an amount that falls within the range, expressed as error percentage, assigned to that error bin. The whole set of error bins covers the entire range of error percentages, with the last one being associated with an open interval (greater than a certain error percentage value). The number of bins, as well as the error ranges associated with the bins are set by the designer before the tuples get placed in them.

In what remains of this section, we explain the issues and techniques behind each step of our idea.

2.1 Chunking and modeling

The first step is to select chunks of the core cuboid that will be modeled. Of course, this assumes that we have computed the core cuboid, from the fact and dimension tables. There are efficient algorithms to compute data cubes (e.g., [18, 2]), however, since we are only interested in computing the core cuboid, a it is sufficient to sort the fact table, and do a pass over the sort data to compute aggregates.

In dividing the core cuboid, the target is to find chunks that are dense enough, so the models have a better chance of fitting the data. A simple way of doing this is to divide the core cuboid space with a regular grid, identifying the chunks

that have potential for being modeled, either because they are dense enough or because sub-chunks contained inside them are dense enough. A pass over the core cuboid data is required to produce the chunks. (We are currently experimenting with techniques to "re-shuffle" the order of the rows in some (or all) dimensions to increase the density of regions.)

At the end of this step, we have classified the chunks into three types: Dense chunks, which are candidates for modeling; Empty chunks, i.e., sub-spaces that do not contain any data cells and can be discarded; Sparse chunks, which contain very few cells. Only the dense chunks are modeled. Computing the models is a step whose implementation depends, of course, on the choice of models. Many different models can be used for cube compression [5] and can be used to implement on-line aggregation. Details of how to efficiently model chunks of data using linear regression can be found in [7]; for loglinear models, see [9]; for kernel estimations see [17], and for wavelets [19]. The choice of the model is orthogonal to our technique affecting only the population distribution of tuples among the error bins.

To illustrate the meaning of models, we present a linear regression model [20] for two dimensions, shown in Equation 1. Each cell value y_{ij}, is shown as being composed by three terms, where c_j is the marginal sum for column j in and r_i is the marginal sum for row i. (The marginal sum is the sum of the values belonging to the particular row or column.) With this method, each cell value would be described by three values: b_0, b_1, b_2,. Notice that we build one of such models for each (dense) chunk in the core cuboid.

$$\hat{y_{ij}} = b_0 c_j + b_1 r_i + b_2 \tag{1}$$

The difference between the estimated value $\hat{y_{ij}}$ and the real value of the cell, y_{ij}, gives us the estimation error, which can be expressed as a percentage of the value y_{ij}, and used to classify the cell to one of the error bins.

It is worth to point out that the modeling of the chunks requires no aditional pass over the core cuboid, since the models can be computed as soon as the chunk is deemed dense enough, and while the data is still in main memory. This, of course, assumes that the cells in the chunk fit in memory. If that is not the case (i.e., we are in the prescence of a big chunk), then some I/O is required to drive the modeling process. Most likely, though, the modeling step will be CPU intensive, and its running time will be determined by the complexity of the models we are trying to build.

The dense chunk descriptions are organized as blocks of data containing the model description, parameter values, and index (we index either non-empty cells or empty cells, depending on which set is smaller). These descriptions are smaller than the data they characterize and thus, many of them can be brought to main memory at once. Indexing of the chunks can help in quickly finding which chunks are needed to answer a query.

2.2 Placing tuples into error bins

As the chunks are being modeled, the error between the estimated value of a cell and its real value can be computed. Thus, the cell can be classified as belonging to an error bin. (The error ranges of the error bins have been previously established.) However, since we do not put cells into error bins, but rather tuples (as we are interested in supporting online aggregation in ROLAP), we still have to find the tuples that contribute to that particular cell in the core cuboid (recall that each tuple in the fact table contributes only to one cell in the core cuboid). We do that by querying the fact table to find the tuples that match the coordinates of the particular cell. Once these tuples are found, they are written in the table associated with the error bin to which the cell has been classified. (The error bin is implemented by a relational table that contains the tuples belonging to that bin.) Tuples corresponding to cells in the sparse chunks are placed directly into the highest-error bin.

At the end of this process, we are left with a set of relational tables, each representing an error bin. (Each containing tuples that contribute to cells whose estimation error lies within the range of the corresponding bin.)

It is important to provide a way to add new tuples to the fact table after the original set of tuples has been placed in the error bins. In our technique this can be acomplished in a simple manner, but we leave the details out of this paper for lack of space (they can be found in [9]).

2.3 Query processing

Given a query posed by a user, the evaluation plan is as follows.

- First we identify the chunks that contain target cells for the query, and use the models to estimate the values of these cells. At the same time (concurrently) we issue a request to the data base engine to fetch the target tuples in the first (highest-error) bin. The first estimate of the answer can be computed with the model estimates This is true for all cells with the exception of those that belong to sparse chunks whose tuples have been stored in the highest-error bin. This fact argues for making the density threshold to decide which chunks are dense enough as small as possible, however, we have to keep in mind that better models are found when the density is higher.
- After the target tuples are retrieved from the highest-error bin, they are aggregated according to the query and combined with the previous answer to give a refined view of the answer to the user. Then more tuples are requested from the next (in error descendent order) bin, until either the user decides to stop the process or all the data has been processed.

Notice that the query processing of the bins requires a data base engine, which could be propietary or an off-the-shelf, commercial data base. The bins are simply relations over which the query is posed. Figure 1 shows a diagram of the architecture of the system.

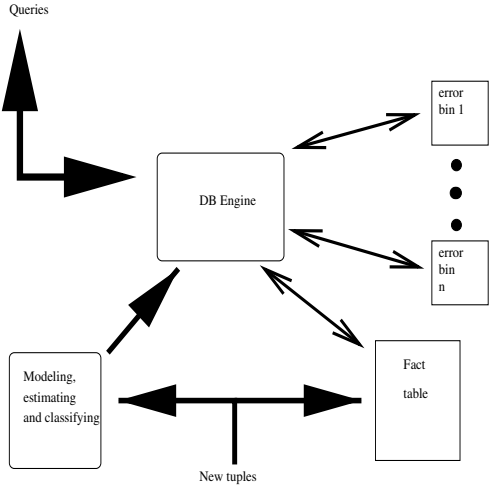


Figure 1: Architecture of the online aggregation system.

3 Experimental Results

Our results were obtained using a real data set which contains data from a survey of retailers. From each tuple of the dataset seven attributes were used (the rest of the data in the tuple is textual). The first six are dimension attributes (such as product, store) and the seventh one is the aggregate attribute (i.e., sale amount). The corresponding cardinalities of the attributes are 597, 42, 966, 1764, 4, 69. The size of each tuple is 102 bytes and the data set has a total of 764,993 tuples (i.e., 78,029,286 bytes). The complete cube for this data set would have 11,792,568,586,176 cells, but since the data is sparse only 335,324 cells are non-zero. We set up six error bins whose error ranges were (in descending order): $\geq 50\%, [40\%, 50\%), [30\%, 40\%), [20\%, 30\%), [10\%, 20\%), < 10\%$. We used Oracle8 Personal Edition as the data base engine, and the experiment was ran on a machine with a Pentium CPU running at 300Mhz. The cube was divided in equally-sized chunks. All the chunks were considered dense (no sparse chunks kept).

Figure 2 shows the time needed to populate the error bins, T_{error} , using the retailer data set. This time includes building the core cuboid, dividing the cube in chunks, modeling the chunks and populating the tables. The third column list the time to build the core cuboid, $T_{2_i core}$ (without modeling regions). The first row in the table corresponds to the experiment we ran using the retailer data set. For the second row, we took only a subset of the data set, corresponding to the first value of attribute 1 and built a smaller Quasi-Cube (and cube) of the dimensions shown in the table. The overhead imposed by the regression method, although considerable (61:1) is manageable: a Quasi-Cube of the larger size ($597 \times 42 \times 966 \times 1764 \times 4 \times 69$) takes approximately 1.5 hours to be built in its entirety. We also see that the time to build Quasi-Cubes scales well with

the size and dimensions of the dataset. The time to build the small Quasi-Cube is not exactly $\frac{1}{597}$ of the time needed to build the larger one, simply because the region we chose contains less than the average number of non-zero cells of regions in the larger cube (286 non-zero cells were present in the smaller cube, while the average per region in the larger cube is 561.68). The process of building Quasi-Cubes is CPU bound. Only 45.22 seconds were spent waiting for I/O when building the large Quasi-Cube. The system does not require anymore I/O than the necessary to bring the relation (in steps) to memory. Figure 3 show the sizes of the error bins and the entire relation.

Dimensions	T_{error}	T_{core}
$597 \times 42 \times 966 \times 1764 \times 4 \times 69$	4,580 sec	74.84 sec
$1 \times 42 \times 966 \times 1764 \times 4 \times 69$	6.0 sec.	0.1 sec.

Figure 2: Time to populate error bins and build core cuboid

Error bin						Fact
1	2	3	4	5	6	table
94,585	86,556	137,136	95,947	112,438	238,337	764,993

Figure 3: Sizes of the error bins and fact table.

Figure 4 show the times needed to retrieve the target data from each one of the error bins when using online aggregation and the time to retrieve the tuples from a single fact table, with no online aggregation support. The first column shows the percentage of data retrieved by the query. The time to present the first answer by using the data models is not reported in the tables, since it is several orders of magnitude smaller than any of the other values. (Computing estimates and errors for a million cells takes roughly 23 seconds of CPU time.)

The results show the benefits of online aggregation. Evaluating the query in the first error bin (highest error) takes almost an order of magnitude less time than waiting for the query to be evaluated in the entire fact table. Hence, the user can have a quick, first answer within a fraction of the time that it takes to evaluate the entire query. As expected, the query evaluation times shown in Figure 4 are proportional to the error bin sizes (and the size of the entire fact table). It is important to point out that for many queries, the initial estimated error is going to be much less than that of an individual cell, since experiments show that aggregating cell values tends to diminish the error considerably.

Percentage	Time to retrieve target tuples from err. bin (sec.)						Time without online aggregation (sec.)
5	3.4	3.24	5.90	3.90	4.60	13.07	25.92
10	3.51	3.30	5.99	3.95	4.62	13.13	26.42
15	3.57	3.35	5.93	4.06	4.62	13.4	27.03
95	4.72	4.12	7.58	4.83	5.82	15.54	35.6
100	4.83	4.12	7.31	4.09	5.77	16.04	36.14

Figure 4: Results of online aggregation.

4 Conclusions

In this paper we have presented a technique that enables online processing of data cube queries. It does so by distributing the tuples in the fact table into a series of tables, or error bins, each corresponding to a level of error incurred when estimating the values of the cells in the core cuboid to which these tuples contribute. The estimations are computed by using models built over chunks of the data cube. These models can be drawn from a variety of techniques, such as linear regression, loglinear models, and wavelets.

We have shown via experiments that the technique is capable of producing quick, yet approximate answers to the queries, reducing the latency normally experienced by queries posed to the fact table directly. The answer gets refined over time as the user is looking at it, and the process can be stopped at will.

We are currently experimenting with the usage of other model than the one reported in this paper (linear regression), as well as considering ways to bound error estimates for aggregations (not individual cells) more tightly, so the initial and subsequent error levels presented to the user are even smaller than the ones that can be reported by the current implementation.

References

1. S. Acharya, P.B. Gibbons, V. Poosala, and S. Ramaswamy. Join Synopses for Approximate Query Answering. In *Proceedings of the 1999 ACM-SIGMOD International Conference on Management of Data, Philadelphia, PA*, June 1999.
2. S. Agarwal, R. Agrawal, P.M. Deshpande, A. Gupta, J.F. Naughton, R. Ramakrishnan and S. Sarawagi. On the Computation of Multidimensional Aggregates, In *Proceedings of the 22nd International Conference on Very Large Data Bases, Bombay, India*, pp. 506-521, September, 1996.
3. A. Agresti, An Introduction to Categorical Data Analysis. John Wiley, New York, 1996.
4. D. Barbará. Quasi-Cubes: Exploiting Approximations in Multidimensional Data Sets. http://www.ise.gmu.edu/ dbarbara/quasi.html.
5. D. Barbará, W. DuMouchel, C. Faloutsos, P.J. Haas, J.M. Hellerstein, Y. Ioannidis, H.V. Jagadish, T. Johnson, R. Ng, V. Poosala, K.A. Ross, and

K.G. Sevcik. The New Jersey Data Reduction Report. *Bulletin of the Technical Committee on Data Engineering*, 20(4):3–45, December 1997.
6. D. Barbará and M. Sullivan. Quasi-Cubes: A space-efficient way to support approximate multidimensional databases. Technical Report, Department of Information and Software Systems Engineering, George Mason University, 1997.
7. D. Barbará and M. Sullivan. Quasi-cubes: Exploiting approximations in multidimensional databases. *SIGMOD Record*, 26(3), September 1997.
8. D. Barbará and X. Wu. Using Approximations to Scale Exploratory Data Analysis in Datacubes. In *Proceedings of the 1999 ACM-SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, CA*, August 1999.
9. D. Barbará and X. Wu. Using loglinear models to compress data cubes. Technical Report, Department of Information and Software Systems Engineering, George Mason University, 1999.
10. C.M. Chen and N. Roussopoulos. Adaptive Selectivity Estimation Using Query Feedback. In Proceedings of the ACM-SIGMOD International Conference on Management of Data, Minneapolis, Minnesota, May, 1994.
11. G. Colliat. OLAP, Relational, and Multidimensional Database Systems. *SIGMOD Record*, 25(3), pp. 64-69, Sep. 1996.
12. J. Gray, A. Bosworth, A. Layman, and H. Pirahesh. Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals. In *Proceedings of the International Conference on Data Engineering, New Orleans*, 1996.
13. V. Harinarayan, A. Rajaraman, and J.D. Ullman. Implementing Data Cubes Efficiently. In *Proceedings of the ACM-SIGMOD Conference, Montreal, Canada*, 1996.
14. P.J. Haas, and J.M. Hellerstein. Ripple Joins for Online Aggregation. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Philadelphia, PA*, May 1999.
15. J.M. Hellerstein, P.J. Haas, and H.J. Wang. Online Aggregation. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Tucson, Arizona*, May 1997.
16. R. Kimball. The Data Warehouse Toolkit: How to Design Dimensional Data Warehouses. John Wiley, New York, 1996.
17. J. Shanmugasundaram, U. Fayyad, and P.S. Bradley. Compressed Data Cubes for OLAP Aggregate Query Approximation on Continuous Dimensions. In *Proceedings of the ACM-SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, CA*, August 1999.
18. D. Srivastava and K. Ross. Fast Computations of Sparse Cubes. In *Proceedings of the 23rd International Conference on Very Large Data Bases, Athens, Greece*, August 1997.
19. J.S. Vitter and M. Wang. Approximate Computation of Multidimensional Aggregates of Sparse Data Using Wavelets. In *Proceedings of the 1999 ACM-SIGMOD International Conference on Management of Data, Philadelphia, PA*, June 1999.
20. R.J. Wonnacott and T.H. Wonnacott. Introductory Statistics. John Wiley, New York,1985
21. P. Young. Recursive estimation and time-series analysis. Springer-Verlag, New York, 1984.

Optimal Multidimensional Query Processing Using Tree Striping

Stefan Berchtold[1], Christian Böhm[2], Daniel A. Keim[3], Hans-Peter Kriegel[2] and Xiaowei Xu[4]

[1] stb gmbh, Ulrichsplatz 6, 86150 Augsburg, Germany
{Stefan.Berchtold}@stb-gmbh.de
[2] University of Munich, Oettingenstr. 67, 80538 Munich, Germany
{boehm,kriegel}@informatik.uni-muenchen.de
[3] University of Halle-Wittenberg, Kurt-Mothes Str. 1, 06099 Halle (Saale), Germany
keim@informatik.uni-halle.de
[4] Siemens AG, Information and Communications, 81730 Munich, Germany
Xiaowei.Xu@mchp.siemens.de

Abstract. In this paper, we propose a new technique for multidimensional query processing which can be widely applied in database systems. Our new technique, called tree striping, generalizes the well-known inverted lists and multidimensional indexing approaches. A theoretical analysis of our generalized technique shows that both, inverted lists and multidimensional indexing approaches, are far from being optimal. A consequence of our analysis is that the use of a set of multidimensional indexes provides considerable improvements over one *d*-dimensional index (multidimensional indexing) or *d* one-dimensional indexes (inverted lists). The basic idea of tree striping is to use the optimal number *k* of lower-dimensional indexes determined by our theoretical analysis for efficient query processing. We confirm our theoretical results by an experimental evaluation on large amounts of real and synthetic data. The results show a speed-up of up to 310% over the multidimensional indexing approach and a speed-up factor of up to 123 (12,300%) over the inverted-lists approach.

1. Introduction

The problem of retrieving all objects satisfying a query which involves multiple attributes is a standard query processing problem prevalent in any database system. The problem especially occurs in the context of feature-based retrieval in multimedia databases [3], but also in relational query processing, e.g. in a data warehouse. The most widely used method to support multi-attribute retrieval is based on indexing the data which means organizing the objects of the database into pages on secondary storage. There is a variety of index structures which have been proposed for this purpose. One of the most popular techniques in commercial databases systems is the inverted-lists approach. The basic idea of the *inverted-lists approach* is to use a one-dimensional index such as a B-tree [4] or one of its variants for each attribute. In order to answer a given range query with *s* attributes specified, it is necessary to access *s* one-dimensional indexes and to perform a costly merge of the partial results obtained from the one-dimensional indexes. For queries involving multiple attributes, however, the merging step is prohibitively expensive and is the major drawback of the inverted-lists approach. *Multidimensional index structures* have been developed as an efficient alternative approach for multidimensional query processing. The basic idea of multidimensional index structures such as space-filling curves [14, 8], grid-file based methods [13, 5], and R-tree-based methods [7, 2], is to use one multi-attribute index which provides efficient access to an arbitrary combination of attributes.

Y. Kambayashi, M. Mohania, and A M. Tjoa (Eds.): DaWaK 2000, LNCS 1874, pp. 244–257, 2000.
© Springer-Verlag Berlin Heidelberg 2000

It is well-known that multidimensional index structures are very efficient for databases with a small number of attributes and outperform inverted lists if the query involves multiple attributes [11]. In many real-life database applications, however, we have to handle databases with a large number of attributes. For databases with a larger number of attributes, the performance of traditional multidimensional index structures rapidly deteriorates. Therefore, specific index structures for high-dimensional data have been proposed. Examples include the TV-tree [12], SS-tree [19], and X-tree [1]. For high dimensions (larger than 12), however, even the performance of specialized high-dimensional index structures decreases.

In this paper, we propose a new approach, called *tree striping*, for an efficient multiattribute retrieval. The basic idea of tree striping is to divide the data space into disjoint subspaces of lower dimensionality such that the cross-product of the subspaces is the original data space. The subspaces are organized using an arbitrary multidimensional index structure. Tree striping is a generalization of the inverted lists and multidimensional indexing approaches, which may both be seen as the extreme cases of tree striping.

The rest of this paper is organized as follows: Section 2 introduces the basic idea of tree striping including the algorithm necessary for processing queries. In Section 3, we then provide a theoretical analysis of our technique and show that optimal query processing is obtained for tree striping. We also show that optimal tree striping outperforms the traditional inverted lists and multidimensional indexing methods. In Section 4, we then discuss the more elaborate query processing algorithms which make use of the specific advantages of "striped" trees and therefore further improve the performance. Section 5 provides the details of our experimental evaluation which includes comparisons of tree striping to inverted lists and two multidimensional index structures, namely the R-tree and the X-tree. The results of our experimental analysis confirm the theoretical results and show substantial speed-ups over the multidimensional indexing and the inverted-lists approaches.

2. Tree Striping

Our new idea presented in this paper is to use the benefits of both the inverted lists and high-dimensional indexing approaches in order to achieve an optimal multidimensional query processing. Our approach, called *tree-striping*, generalizes both previous approaches.

2.1 Basic Idea

The basic idea of tree-striping is to divide the data space into disjoint subspaces of lower dimensionality such that the cross-product of the subspaces is the original data space[1]. This means that each subspace contains a number of attributes (dimensions) and each object of the database occurs in all subspaces. For example, the three-dimensional data space (customer_no, discount, turnover) may be divided into the one-dimensional subspace (customer_no) and the two-dimensional subspace (discount, turnover). Obviously, the dimensionality of the subspaces is smaller than the dimensionality of the data space, and hence, we are able to index the subspaces more efficiently using any multidimensional index structure.

To insert an object, we divide the object into subobjects according to the division of the data space. Then, we insert the subobjects in the multidimensional index structure managing the corresponding subspace. To process a query, we divide the query accord-

1. Note that a division of the data space into disjoint subspaces is different from a partitioning of the data space where the partitions have the same dimensionality as the original data space whereas subspaces have a lower dimensionality.

ing to the division of the data space and issue the subqueries to the relevant multidimensional indexes. In a second step, we merge the results which have been produced by the indexes using an external sorting algorithm such as merge sort. The general idea and query processing strategy of tree striping is presented in Figure 1.

Note that, in contrast to inverted lists, in general, the selectivity of subspace indexes is relatively high because each index manages information about more than one attribute. Therefore, the amount of partial results produced in the first step is rather small which means that the cost for the merging step are not significant. Our formal model, which will be presented in Section 3, confirms this fact.

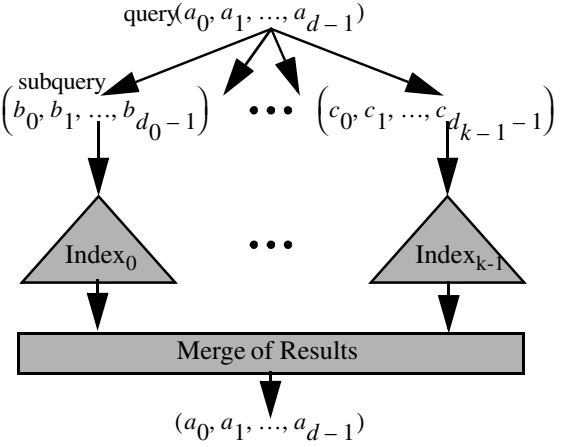


Fig. 1: Tree Striping

It is clear that the number and dimensionality of the data space divisions are important parameters for the performance of our technique. The optimal division mainly depends on the dimension, the number of data items, and the data distribution. The parameters have to be chosen adequately to achieve an optimal performance. For a uniform data distribution, the parameters for an optimal division into subspaces can be obtained easily from the theoretical analysis (cf. Section 3).

2.2 Definition of Tree Striping

In this Section, we formally define the tree striping technique. In the following, we consider objects as vectors in a vector space and attributes as components of the vectors. Given is a data space of dimension d and extension $[0..1]^d$, N vectors v having components $v_0 \ldots v_{d-1}$ and an arbitrary multidimensional index structure *MIS* supporting the relevant query types. First, we need a mapping which assigns the dimensions to the different subtrees.

Definition 1: (Dimension Assignment):

The dimension assignment DA is a mapping $R^d \rightarrow (R^{d_0}, \ldots, R^{d_{k-1}})$ of a d-dimensional vector v to a vector of k d_l-dimensional vectors w^l, $0 \le l < k$, such that the following conditions hold:

1. $\sum_{l=0}^{k-1} d_l = d$

2. $\forall j \;\; 0 \le j < d, \;\; \exists l \;\; 0 \le l < k, \;\; \exists i \;\; 0 \le i < d_l: \;\; v_j = w_i^l$

3. $\forall l \;\; 0 \le l < k, \;\; \forall i \;\; 0 \le i < d_l, \;\; \exists j \;\; 0 \le j < d: \;\; w_i^l = v_j$

Note that w_i^l denotes the i-th component in the l-th index. To clarify the definition of dimension assignment, we provide a simple example: Given a 5-dimensional data space (d=5). We may define a dimension assignment DA_{odd_even} such that $k = 2$, $d_0 = 3$, and d_1

= 2, i.e. DA_{odd_even} divides the data space into two subspaces of dimensionality 3 and 2. Explicitly, DA_{odd_even} maps even dimensions to the first subspace and odd dimensions to the second subspace, more formally, $DA_{odd_even}(v) = (w_0, w_1)$, $w_0 = (v_0, v_2, v_4)$, $w_1 = (v_1, v_3)$. Thus a vector v= (0, 4, 6, 5, 1) is mapped to $DA_{odd_even}((0, 4, 6, 5, 1)) = ((0, 6, 1), (4, 5))$. Obviously, DA_{odd_even} meets the conditions specified in Definition 1 because all dimensions of the data space have been mapped to a subspace and vice versa.

Using the definition of dimension assignment, we are now able to formally define tree-striping:

Definition 2: (Tree Striping):

Given a database DB of N d-dimensional vectors and a dimension assignment DA. Then, a tree-striping TS is defined as a vector of k d_l-dimensional indexes

$$MIS^l = \{w^l\}\ ,\ 0 \le l < k\ , \qquad \text{with } w^l = DA^l(v)\ ,\ v \in DB\ .$$

Tree striping as defined in Definition 2 is a generalization of the previous approaches. For the special case of $k = d$, tree striping corresponds to inverted lists because the dimension assignment produces d one-dimensional data objects; and for the special case of $k = 1$, tree striping corresponds to the traditional multidimensional indexing approach because we have one d-dimensional index. The most important question is whether there exists a tree striping which provides better results than the extremes (the well-known inverted lists and multidimensional indexing approaches). In particular, we have to determine whether there exists a k ($1 < k < d$) such that tree striping outperforms the other approaches. In the next Section, we introduce a theoretical model showing that an optimal k exists. Our experimental analysis presented in Section 5 confirms the results of our theoretical model and shows performance improvements of up to a factor of 120 times over the inverted lists and up to 280% over the multidimensional indexing approach. A second open question is how the attributes (dimensions) are assigned to the different trees such that the performance improvement is optimal. In Section 4, we discuss the implications of different dimension assignments and also introduce optimized algorithms for query processing using striped trees.

```
SetOfObject query(TreeStrip ts, QuerySpec qs)
{
    int i;
    SetOfSubObject sst[ts.num];
    SubQuerySpec sqs[ts.num];
    SetOfObject st;

    // for all indexes
    for (i = 0; i < ts.num; i++)
    {
        // query i-th index with sub-query
        sqs[i] = ts.opt_dim_assign(i, qs);
        sst[i] = ts.index[i].query(sqs[i]);
        // sort result by primary key
        sst[i].sort();
    }
    // now merge single results
    st = merge(sst, ts.num);
    return st;
}
```

Fig. 2: A First Query Processing Algorithm

Note that tree striping as defined so far is independent of the multidimensional index structure used. Any multidimensional index structure such as the R-tree [7] and its variants (R+-tree [17], R*-tree [2], P-tree [9]), Buddy-tree [16], linear quadtrees [6], z-ordering [14] or other space-filling curves [8], and grid-file based methods [13, 5] may be used for this purpose.

Before we describe our theoretical model, we first provide a simple algorithm for processing queries using striped trees. As the single indexes do not have all information about an object, but only about some attributes of the object, in general, we have to query all indexes in order to process a query. We therefore divide the query specification *qs* into sub-

query specifications *sqs*[*l*] according to the dimension assignment. Then, we query each single index with the sub-query specification *sqs*[*l*] and record the results. In a final step, we have to merge the results by sorting the single results according to the primary key of the objects or any object identificator. Figure 2 shows a first version of a query processing algorithm. An optimized version for querying striped trees is provided in Section 4.

3. Theoretical Model

As already mentioned, most of the multidimensional indexing approaches efficiently solve the multi-attribute retrieval problem on low-dimensional data. From our experience, in real-life database projects, we have learned that even for relational database systems, handling relatively high numbers of attributes (more than 10) occurs, for which the performance of traditional index structures deteriorates. To process arbitrary queries (e.g., point, range, and partial match queries) efficiently on those databases, we have to equally index all the attributes which means that we have to deal with a high-dimensional data space.

Unfortunately, some mathematical problems arise in high-dimensional spaces which are usually summarized by the term 'curse of dimensionality.' A basic effect in high-dimensional space is the exponential growth of the volume: Let us assume a database of 1,000,000 uniformly distributed objects consisting of 20 numerical attributes in the range [0...1]. Let us further assume that we are interested in a query which provides 10 result objects located around the midpoint of the data space (0.5, 0.5, 0.5, ..., 0.5). Which range do we have to query in order to obtain 10 result objects? Obviously, we have to assure that the volume of our query range equals to $10/1{,}000{,}000 = 10^{-5}$, as the volume of the data space equals to 1. This leads to a query range in each attribute of $\sqrt[20]{10^{-5}} \approx 0.56$. So we have to query the range (0.22-0.78, 0.22-0.78, ..., 0.22-0.78). That means a query with a selectivity of 10^{-5} leads to a query range of 0.56 in each attribute in a 20-dimensional data space.

Considering these effects, we are able to provide a concise cost model of processing range queries in a high-dimensional data space using the tree striping technique. For the following, we assume a uniformly distributed set of N vectors in a d-dimensional space of extension $[0..1]^d$. Note that, although we assume a uniform distribution of the data, our model can be applied to real data as well (cf. Section 5). We will use the cost model to determine the optimal number of trees and accordingly the dimensions of the trees for a given data set, i.e. the optimal dimension assignment.

Our cost model is divided into two parts: First, the cost arising from querying the striped trees, and second, the cost for merging the results of the striped trees into one final result. Both cost functions are highly influenced by the dimensions of the striped trees. The index lookup cost is growing super-linearly with growing tree dimension. However, the merging cost is growing super-linearly with the size of the result, which is, in turn, falling with dimension of the trees. This fact implies the assumption that the total cost could form a minimum where both costs are moderate. This minimum should be located anywhere between the d-dimensional index and the inverted-lists approaches.

Several cost models for queries on multidimensional index structures have been published. The most suitable ones for our purposes are the model of Kamel and Faloutsos [10] and the similar model of Pagel, et.al. [15]. We decided to use the model of Kamel and Faloutsos as a basis for our considerations. We therefore assume that the multidimensional index structure aggregates a fixed number of C_{eff} vectors into a data page

such that the bounding box containing the vectors forms a square-shaped hyper-rectangle with the (hyper-) volume

$$V_{BB} = \frac{C_{eff}(d)}{N} .$$

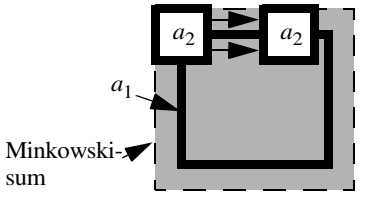


Fig. 3: The Minkowski Sum

Thus, C_{eff} denotes the actual fan-out of the index. From that, the edge length σ of a typical bounding box is

$$\sigma = \sqrt[d]{V_{BB}} = \sqrt[d]{\frac{C_{eff}(d)}{N}} .$$

Next, we determine the probability that such a data page is loaded when processing a square range query with volume V_q. Analogously, we compute the edge length q of V_q as $q = \sqrt[d]{V_q}$. The probability can be determined using the so-called Minkowski sum. Intuitively, the Minkowski sum of two areas a_1 and a_2 can be constructed by painting a_1 using a_2 as a brush (cf. Figure 3). If both areas are multidimensional rectangles, we have to add the side-lengths of a_1 and a_2 accordingly. Thus, the Minkowski sum of query volume and the bounding box is

$$Mink(V_{BB}, V_q) = (\sigma + q)^d = \left(\sqrt[d]{\frac{C_{eff}(d)}{N}} + q\right)^d .$$

The Minkowski sum $Mink(V_{BB}, V_q)$ equals to the probability that a randomly located bounding box and a randomly located query intersect. Thus, the expected number of data pages intersecting V_q is $Mink(V_{BB}, V_q)$ multiplied by the number of pages:

$$P_{\text{index}}(d, N, q) = \frac{C_{eff}(d)}{N} \cdot \left(\sqrt[d]{\frac{C_{eff}(d)}{N}} + q\right)^d = \left(1 + q \cdot \sqrt[d]{\frac{N}{C_{eff}(d)}}\right)^d$$

The number C_{eff} of data vectors in a data page also depends on the dimension d of the vectors. Assuming that each coordinate value is stored as a 32-bit floating point value and that there is an additional unique object identifier which also requires 32 bit, we determine C_{eff} as:

$$C_{eff} = \frac{\text{page-size} \cdot \text{storage-utilization}}{4 \cdot (d + 1)}$$

The cost for combining the results of the multidimensional index accesses mostly depend on the selectivities of the indexes. If |FRS| is the size of the final result set of query Q then $|IRS_i|$ is the intermediate result set produced by the i-th index having dimension d_i. Thus:

$$\frac{|IRS_i|}{N} = \left(\frac{|FRS|}{N}\right)^{\frac{d_i}{d}} = q^{d_i}$$

Note that we have to sort each intermediate result set according to the object identifiers in order to be able to merge them into the final result set. We have to apply an external sorting algorithm since, for larger q or minor d_i, the result set will exceed the available main memory. According to Ullman [18], the cost for performing multiway merge-sort on a relation of B blocks is $2B \cdot \log_M(B)$ where M is the number of cache pages available to the sorting process. We can store the object identifiers in a densely packed fashion such that $|IRS_i|$ object identifiers require $4 \cdot |IRS_i| / \text{page-size}$ pages. From that, the cost for sorting the result set of a single index are:

$$P_{\text{sort}}(d_i, N, q) = \frac{8 \cdot N \cdot q^{d_i}}{\text{page-size}} \cdot \log_M\left(\frac{(4 \cdot N \cdot q^{d_i})}{\text{page-size}}\right)$$

To determine the total cost, P_{sort} and P_{index} have to be summed up for all striped trees. For merging the result sets, each of them has to be scanned once more. Total cost is:

$$P(d_i, N, q) = \sum_{i=1}^{k} \left[\left(1 + q \cdot \sqrt[d_i]{\frac{N}{C_{eff}(d)}} \right)^{d_i} + \frac{4 \cdot N \cdot q^{d_i}}{\text{page-size}} \cdot \left(1 + 2 \cdot \log_M \left(\frac{(4 \cdot N \cdot q^{d_i})}{\text{page-size}} \right) \right) \right]$$

In the following, we assume, that the d dimensions of our data space are striped into k divisions[1]

$$d_0 = d_1 = \ldots = d_{k-1} = \frac{d}{k}$$

In this case, our cost function can be simplified to:

$$P(d, k, N, q) = k \cdot \left[\left(1 + q \cdot \left(\frac{N}{C_{eff}(d/k)} \right)^{k/d} \right)^{d/k} + \right.$$

$$\left. + \frac{4 \cdot N \cdot q^{d/k}}{\text{page-size}} \cdot \left(1 + 2 \cdot \log_M \left(\frac{4 \cdot N \cdot q^{d/k}}{\text{page-size}} \right) \right) \right]$$

Figure 4 shows the total cost over k in a typical setting with a database of 1,000,000 uniformly distributed objects in a 15-dimensional data space. The selectivity of the query is 0.01%. There is a clear minimum between k=2 and k=3.

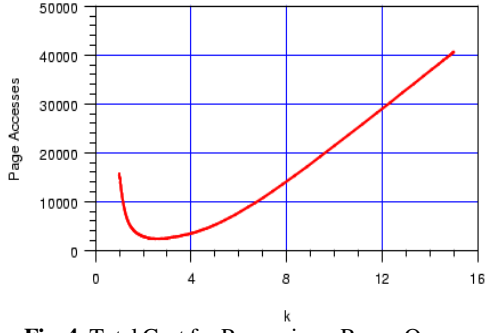


Fig. 4: Total Cost for Processing a Range Query

Thus, we are able to determine an optimal k by solving the following equation:

$$\frac{\partial}{\partial k} P(d, k, N, q) = 0 \quad \text{(eq. 1)}.$$

The analytic evaluation of this equation yields a rather large formula which is omitted due to space limitations. A MAPLE-generated C function determining the derivative can be used to calculate the optimum.

Unfortunately, the cost model presented so far is accurate only in the low-dimensional case. This is caused by the fact that in high-dimensional data spaces the data pages cannot be split in each dimension. If we split a 20-dimensional data space once per dimension, we obtain 2^{20}=1,000,000 data pages. Obviously, the number of data objects would have to grow exponentially with the dimension in order to allow one split per dimension. Therefore, we provide a special high-dimensional adaptation of our cost model. Our extension assumes that data pages are split only in the first d' dimensions where d' is the logarithm of the number of data pages to the basis of two:

$$d' = \log_2 \left(\frac{N}{C_{eff}(d_i)} \right) .$$

The data pages have the average extension 1/2 in d' dimensions and extension 1 in all remaining dimensions (d-d'). When determining the Minkowski sum, we additionally

1. For d_0, ..., d_{k-1}, only whole numbers are meaningful. This effect is handled later, but is of minor importance for our cost model.

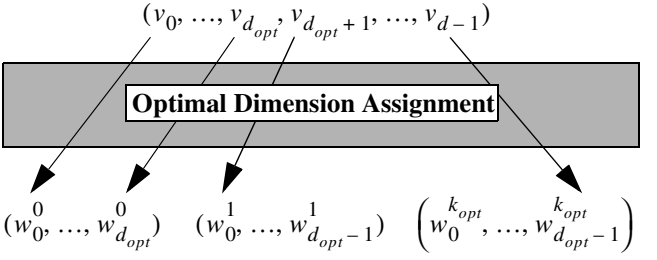


Fig. 5: Optimal Dimension Assignment

have to consider that only a part of the volume is located inside the data space because in the dimensions which have not been split, the extension of the Minkowski-sum is still 1, rather than (1+q):

$$HiDiMink(V_{BB}, V_q) = \left(\frac{1}{2} + \frac{q}{2}\right)^{d'} \cdot 1^{d-d'} \quad .$$

Thus, the expected number of data pages accessed in the high-dimensional case is:

$$P_{\text{index, HiDi}}(d, N, q) = \frac{N}{C_{eff}(d_i)} \cdot \left(\frac{1}{2} + \frac{q}{2}\right)^{\log_2(\frac{N}{C_{eff}(d)})}$$

Adding the sort cost we obtain the following total cost for high-dimensional data spaces:

$$P_{\text{HiDi}}(d, k, N, q) = k \cdot \left[\frac{N}{C_{eff}(d_i)} \cdot \left(\frac{1}{2} + \frac{q}{2}\right)^{\log_2(\frac{N}{C_{eff}(d_i)})} + \right.$$

$$\left. + \frac{4 \cdot N \cdot q^{d/k}}{\text{page-size}} \cdot \left(1 + 2 \cdot \log_M(\frac{4 \cdot N \cdot q^{d/k}}{\text{page-size}})\right) \right]$$

4. Query processing

For optimal response times, we have to make two decision: We first have to choose an adequate dimension assignment and second, we have to choose the right strategy for processing queries.

As a result of the theoretical analysis presented in Section 3 there exists an optimal number k of striped trees, which can be determined according to our cost model (cf. equation 1). Since k is a real number, however, we cannot directly use k as a parameter for our query processor. Instead, we use the floor of k

$$k_{opt} = \lfloor k \rfloor$$

and then determine the optimal dimensionality of our trees given by

$$d_{opt} = \lfloor d/k \rfloor \, .$$

Since in general, $(k_{opt} \cdot d_{opt})$ is smaller than d we have to distribute the remaining

$$d_{rem} = d - (k_{opt} \cdot d_{opt})$$

```
void insert(TreeStrip ts, object t)
{   int l;
    SubObject st[ts.num];
    // for all indexes
    for (l = 0; l < ts.num; l++)
    {   // determine sub-objects
        st[l] = ts.opt_dim_assign(l, t);
        // insert sub-objects into l-th index
        ts.index[l].insert(st[l]);
    }
}
```

Fig. 6: Insertion Algorithm

attributes to our trees. Thus, we obtain d_{rem} trees with dimensionality $(d_{opt}+1)$ and $(k_{opt}-d_{rem})$ trees with dimensionality d_{opt}. In the following, we have to distinguish between two cases: The first case is that we have additional information about the selectivity of the attributes, which usually occurs for relational databases. The second case is that we have no additional information which usually occurs in indexing multimedia data using feature vectors. Let us first consider the more general case that we do not have any additional information and therefore assume that all attributes have the same selectivity. In this case, the optimal dimensionality d_{opt} of our trees may be used to define the following Optimal Dimension Assignment.

Definition 3: (Optimal Dimension Assignment):
The dimension assignment DA_{opt} is a dimension assignment according to Definition 1 such that:

$$DA^l(v)_i = w_i^l = \begin{cases} v_{l(d_{opt}+1)+i} & \text{if } l < d_{rem} \\ v_{d_{rem}(d_{opt}+1)+(l-d_{rem})d_{opt}+i} & \text{otherwise} \end{cases},$$

where $0 \le l < k_{opt}$, $d_l = \begin{cases} d_{opt}+1 & \text{if } l < d_{rem} \\ d_{opt} & \text{otherwise} \end{cases}$, and $0 \le i < d_l$.

Intuitively, the optimal dimension assignment assigns the i-th component of the original vector v to a component of one of the vectors w^l such that the first vector w^0 receives the first d_0 components $(v_0 \ldots v_{d_0-1})$, the second vector w^1 accommodates the components $(v_{d_0} \ldots v_{d_0+d_1-1})$ and so on.

Using the optimal dimension assignment according to Definition 3, we now are able to present the insert algorithm of our tree striping technique, as depicted in Figure 6. In order to insert an object t, we simply divide t into a set of k_{opt} sub-objects $st[l]$ (using the optimal dimension assignment) and insert them into the according striped tree $ts.index[l]$ $(0 \le l < k_{opt})$.

A more complex algorithm is required for processing queries on striped trees. A rather simple query processing algorithm has already been presented in Section 2. The algorithm depicted in Figure 2, however, has a major drawback: Let us assume that we have to process a partial range query *PRQ* which specifies attributes a, b and c:

$$PRQ = \{*,*,[a_l, a_u], *,*,[b_l, b_u], *,*,[c_l, c_u], *,*\} .$$

Let us further assume that all these three attributes are located in the first of the striped trees. Obviously, it does not make sense to query any tree other than the first tree because all other trees do not have any selectivity. The algorithm presented in Figure 2, however, executes queries on all trees ignoring the expected selectivity of the trees. In order to process queries efficiently we have to take the selectivity of a tree into account and query a tree only if the expected gain in selectivity is worth the cost of querying the tree.

Another potential improvement of the query processing algorithm can be exemplified by the following situation: Assume that the three specified attributes a, b and c in the

above example are spread over two striped trees T_0 managing attributes a and b, and T_1 managing attribute c. After querying tree T_0 we will typically receive a set of answers (candidates) which may contain some false hits. This assumption holds because the selectivity of T_0 is much higher than the selectivity of T_1. If we furthermore assume to have meaningful queries, i.e. queries having a good selectivity on all attributes, in general the set of candidates will be small. In this case, the cost for loading the candidate objects from secondary storage and checking if the objects fulfill the query specification may be lower than the cost of querying additional trees.

```
SetOfObject query(TreeStrip ts, QuerySpec qs)
{
    int i, cost_index, cost_linear;
    SetOfSubObject sst[ts.num];
    SubQuerySpec sqs[ts.num];
    SetOfObject st; // set of candidates
    // sort indexes according to selectivity
    ts.sort_index(qs);
    // determine sub-queries
    for (i = 0; i < ts.num; i++)
        sqs[i] = ts.opt_dim_assign(i, qs);
    i = 0;
    // estimate cost
    cost_index = cost_modell(sqs[0]);
    cost_linear = cost_linear_scan(sqs[0]);
    while (i < ts.num &&
            cost_index < cost_linear)
    {   // query index
        sst[i] = ts.index[i].query(sqs[i]);
        // sorted merge of result
        sst[i].sort();
        merge(st, sst, ts.num);
        // estimate cost
        if (i < ts.num)
        {   cost_index = cost_modell(st, sqs[i+1]);
            cost_linear = cost_linear(st);
        }
    }
    if (i < ts.num)
    {   // load attributes
        database.load(st);
        remove_false_hits(st, qs);
    }
    return st;
}
```

Fig. 7: Query Processing using Tree Striping

Let us now consider the second case where we do have some additional information about the selectivity of the attributes. A different selectivity of the attributes may be induced by the attributes of different data types (e.g., a boolean attribute usually has a selectivity of 50%) and by different data distributions. We can use this information to adapt the optimal dimension assignment. If we are able to query the tree containing the attributes with the highest selectivity first, the resulting set of candidates will be rather small and will contain only a few false hits. Therefore, query processing can be finished without querying the other trees. This means that, if we have information about the selectivity of attributes, we should sort the attributes according to their selectivity before applying the dimension assignment[1]. Note that in some cases, a non-uniform division may lead to better results. For example, let us assume that we have objects with 9 attributes (a, b, ... i), that k_{opt} equals to 3, and that the attributes a to d have a high selectivity whereas the selectivity of attributes e to i is rather low. Then, it is beneficial to divide the objects into sub-objects (a, b, c, d), (e, f), and (g, h, i) which would be a sub-optimal division assuming no a-priori knowledge about the selectivity of attributes.

Considering all these effects, we are now able to provide a more sophisticated algorithm for the processing of queries on striped trees. The algorithm (cf. Figure 7) first determines whether a linear search of the database is expected to be cheaper than a search using trees which may be the case for very large queries. The algorithm then sorts the striped trees according to their selectivity, i.e. the tree which probably provides the

1. Note that this operation involves not only the query-processing but also the dimension assignment, since we have to ensure that the attributes with the best selectivity are assigned to the first trees.

smallest set of candidates is queried first. If the querying of the first tree leads to a small set of candidates, we determine whether loading these candidates from secondary storage is cheaper than querying the second tree. If this is the case, we load the attributes and output all candidates fulfilling the query specification. Otherwise, we query the second tree. This process iterates until all trees have been queried or the candidates are loaded and processed.

As the implementation of multidimensional index structures is complex, the assignment of different data types such as strings and floating numbers into one tree is not practicable. The division of a object may therefore be induced not only by the expected performance improvement but also by other considerations. Obviously, this can lead to sub-optimal dimension assignments. Our practical experience, however, shows that a slightly sub-optimal dimension assignment performs nearly as well as the optimal dimension assignment.

5. Experimental Analysis

To show the practical relevance of our method, we performed an extensive experimental evaluation of tree striping and compared it to the inverted lists and the multidimensional indexing approach. All experimental results have been computed on an HP9000/780 workstation with several GBytes of secondary storage. For the experiments, we used an object-oriented implementation (C++) of the R*-tree [2] and the X-tree [1].

The test data used for the experiments are real data consisting of text data describing substrings of a large database of texts, and synthetic data consisting of uniformly distributed points in high-dimensional space. The block size used for our experiments is 4 KByte, and all query processing techniques were allowed to use the same amount of cache. For a realistic evaluation, we used very large amounts of data (up to 80 MBytes) in our experiments. The total amount of disk space occupied by the created indexes (inverted lists, multidimensional indexes and tree-striped indexes) is about 2 GBytes and the CPU time for inserting the data adds up to about one week.

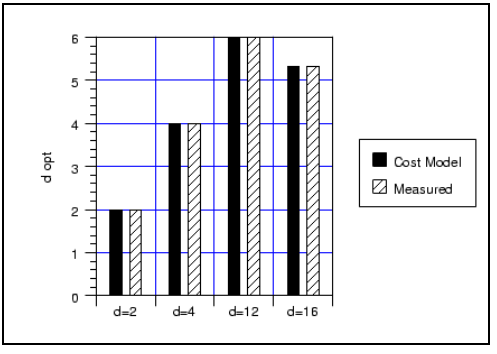
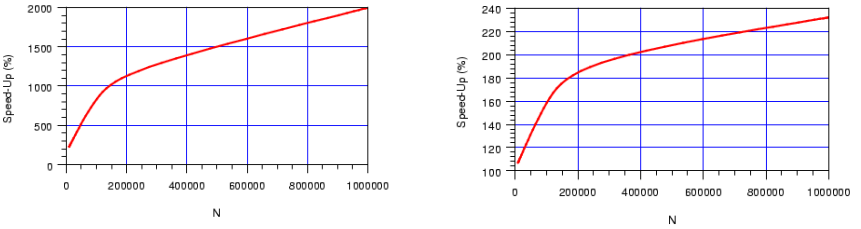


Fig. 8: Comparison of Measured Optimal Dimension Assignment and Model Estimation

In a first experiment, we confirmed our theoretical result (cf. Section 3) that the tree striping technique as a generalization of the lists and multidimensional indexing approaches outperforms both other techniques. For the experiment, we used 1,000,000 uniformly distributed data objects of varying dimensionality ($d = 2..16$). We built the according indexes (R*-tree) and queried the indexes with a selectivity of 10^{-5} which corresponds to an expected result of about 10 hits. In order to avoid statistical effects, we used the average cost of 100 uniformly distributed query windows. The observed variance was rather small. We compared different tree stripings (varying the value of k) and determined the optimal dimension assignment (optimal value of k). The tested dimension assignments for the 16-dimensional data set are (16), (8, 8), (6, 5, 5), (4, 4, 4, 4), (2, 2, 2, 2, 2, 2, 2, 2), and (1, 1, ..., 1, 1). The data sets of other dimensionality have been tested analogously. In



a. Speed-Up over Inverted Lists **b. Speed-Up over Multidimensional Indexing**

Fig. 9: Speed-Up of Tree Striping for an Increasing Number of Data Items

Figure 8, we show the optimal dimensionality *(d_{opt})* of striped trees depending on dimensionality of the data. For d=2 and d=4, the optimal dimension assignment of tree striping provides one d-dimensional index, i.e. it is identical to multidimensional indexing. As expected according to our theoretical analysis, for higher dimensions the optimal dimension assignment of tree striping is between the extreme cases: For d=12, we obtain two 6-dimensional indexes and for d=16, we obtain a division into 3 indexes with dimensionality (6, 5, 5). Note that in all experiments, the optimal dimension assignment estimated by our cost model exactly matches the measured optimum. For our experiments, we therefore use the optimal dimension assignment as determined by our cost model.

Another important criterion for the evaluation of indexing techniques is their scalability, that is the behavior of the technique for an increasing size of the database. We therefore performed an experiment using a fixed dimensionality (d=16) and a fixed query selectivity of 10^{-5} and varied the number of data items from 10,000 to 1,000,000. Again, we used our cost model to determine the optimal dimension assignment. The speed-up over multidimensional indexing starts with a moderate value of 107% for a small database but, as the size of the database increases, the speed-up also increases to up-to 230% over multidimensional indexing for the largest database of 1,000,000 objects (cf. Figure 9). The speed-up over the inverted-lists approach starts with 228% and reaches its maximum of 2,000% (20 times faster) for the largest database of 1,000,000 objects (cf. Figure 9).

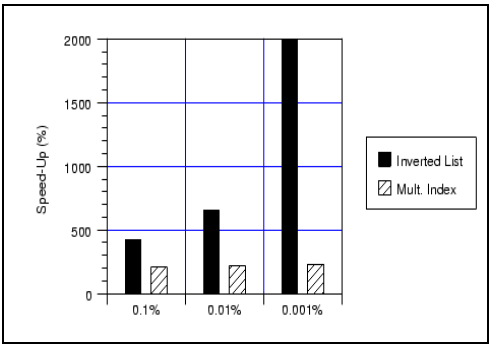


Fig. 10: Performance for Varying Selectivities

The intention of the experiment depicted in Figure 10 is to show that the high speed-ups are independent from the selectivity of the queries. We repeated the previous experiments for different dimensionality (shown are the experiment for d=12 and d=16) using selectivities between 10^{-3} and 10^{-5}. Again, we obtained a speed-up of 210% to 220% over the multidimensional index and a speed-up factor of 4 to 20 over the inverted lists.

To show the practical relevance of our technique, in a last series of experiments, we evaluated the tree striping technique using real data which consists of text data describing substrings of a large database of texts.

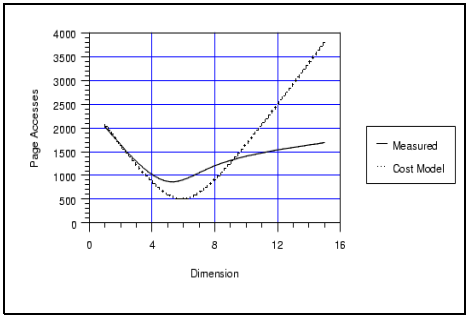


Fig. 11: Optimal Dimension Assignment for Real Data (Text Data)

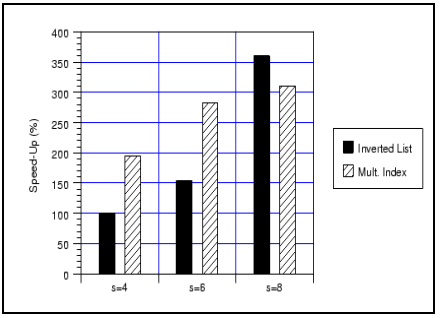


Fig. 12: Performance of Partial Range Queries (Text Data)

In Figure 11, we compare the measured performance for range queries with a selectivity of 0.2% to the performance determined by our model (cf. Section 3). The minima of the two curves correspond to the optimal dimension assignment (d_{opt}). Note that the model estimates the optimal dimension assignment correctly ($d_{opt} \approx 5$) although it assumes a uniform distribution of the data. The difference between model and measurements for large dimensions (i.e. small k), however, may be explained by the non-uniform distribution of the real data. In Figure 12, we present the speed-up of tree striping over inverted lists and multidimensional indexing for partial range queries with a varying number of attributes specified (s=4..8). It is interesting that for a partial range query with 4 attributes specified, tree striping degenerates to inverted lists. If more than 4 attributes are specified, tree striping becomes better than both, inverted lists and multidimensional indexing. Note that for s=6, inverted lists are better than multidimensional indexing, whereas for s=8, multidimensional indexing is better than inverted lists.

6. Conclusions

In this paper, we propose a new technique for multidimensional query processing, called tree striping. Tree striping is a generalization of the inverted-lists technique and the multidimensional indexing approach. A theoretical analysis of our technique shows that tree striping clearly outperforms both - the inverted lists and multidimensional indexing approaches. An experimental evaluation of our technique confirms the results of our theoretical analysis, unveiling significant speed-up factors of tree striping over inverted lists and multidimensional indexing for different databases of varying size and dimensionality, as well as for different query types.

Our future work will include an application of tree striping to other multidimensional index structures not handled in this paper and we expect a substantial performance improvement over the non-striped version. We further plan to develop a parallel version of the tree striping technique. We expect a nearly linear speed-up for a parallel version since the separate indexes can be queried independently, which should provide a linear speed-up of the query processing time.

References

1. Berchtold S., Keim D., Kriegel H.-P.: *'The X-tree: An Index Structure for High-Dimensional Data',* 22nd Conf. on Very Large Databases, 1996, Bombay, India.
2. Beckmann N., Kriegel H.-P., Schneider R., Seeger B.: *'The R*-tree: An Efficient and Robust Access Method for Points and Rectangles',* Proc. ACM SIGMOD Int. Conf. on Management of Data, Atlantic City, NJ, 1990, pp. 322-331.
3. Faloutsos C., Barber R., Flickner M., Hafner J., et al.: *'Efficient and Effective Querying by Image Content',* Journal of Intelligent Information Systems, 1994, Vol. 3, pp. 231-262.
4. Bayer R., McCreight E.M.: *'Organization and Maintenance of Large Ordered Indices',* Acta informatica, Vol. 1, No. 3, 1972, pp. 173-189.
5. Freeston M.: *'The BANG file: A new kind of grid file',* Proc. ACM SIGMOD Int. Conf. on Management of Data, San Francisco, CA, 1987, pp. 260-269.
6. Gargantini I.: *'An Effective Way to Represent Quadtrees',* Comm. of the ACM, Vol. 25, No. 12, 1982, pp. 905-910.
7. Guttman A.: *'R-trees: A Dynamic Index Structure for Spatial Searching',* Proc. ACM SIGMOD Int. Conf. on Management of Data, Boston, MA, 1984, pp. 47-57.
8. Jagadish H. V.: *'Linear Clustering of Objects with Multiple Attributes',* Proc. ACM SIGMOD Int. Conf. on Management of Data, Atlantic City, NJ, 1990, pp. 332-342.
9. Jagadish H. V.: *'Spatial Search with Polyhedra',* Proc. 6th Int. Conf. on Data Engineering, Los Angeles, CA, 1990, pp. 311-319.
10. Kamel I., Faloutsos C.: '*On Packing R-trees*', CIKM, 1993, pp. 490-499.
11. Kriegel H.-P.: *'Performance Comparison of Index Structures for Multi-Key Retrieval',* Proc. ACM SIGMOD Int. Conf. on Management of Data, Boston, MA, 1984, pp. 186-196.
12. Lin K., Jagadish H. V., Faloutsos C.: *'The TV-tree: An Index Structure for High-Dimensional Data',* VLDB Journal, Vol. 3, 1995, pp. 517-542.
13. Nievergelt J., Hinterberger H., Sevcik K. C.: *'The Grid File: An Adaptable, Symmetric Multikey File Structure',* ACM Trans. on Database Systems, Vol. 9, No. 1, 1984, pp. 38-71.
14. Orenstein J., : *'A comparison of spatial query processing techniques for native and parameter spaces',* Proc. ACM SIGMOD Int. Conf. on Management of Data, 1990, pp. 326-336.
15. Pagel B.U., Six H.-W., Toben H., Widmayer P.: '*Toward an Analysis of Range Query Performance in Spatial Data Structures*', Proc. ACM PODS, Symposium on Principa1s of Database Systems, 1993, pp. 214-221.
16. Seeger B., Kriegel H.-P.: *'The Buddy Tree: An Efficient and Robust Access Method for Spatial Data Base Systems',* Proc. 16th Int. Conf. on Very Large Data Bases, Brisbane, Australia, 1990, pp. 590-601.
17. Sellis T., Roussopoulos N., Faloutsos C.: 'The R+-Tree: A Dynamic Index for Multi-Dimensional Objects', Proc. 13th Int. Conf. on Very Large Databases, Brighton, England, 1987, pp. 507-518.
18. Ullman J.D.: *'Database and Knowledge-Base System', Vol. II*,Compute Science Press, Rockville, MD, 1989.
19. White, D., Jain R.: *'Similarity Indexing with the SS-tree',* Proc. 12th Int. Conf. on Data Engineering, New Orleans, LA, 1996, pp. 516-523.

Enhancing Preprocessing in Data-Intensive Domains using Online-Analytical Processing

Alexander Maedche[1], Andreas Hotho[1], and Markus Wiese[2]

[1] Institute AIFB, Karlsruhe University, D-76128 Karlsruhe, Germany
{maedche, hotho}@aifb.uni-karlsruhe.de,
http://www.aifb.uni-karlsruhe.de/
[2] Deutsche Telekom AG , D-76646 Bruchsal, Germany
markus.wiese@telekom.de,
http://www.telekom.de/

Abstract The application of data mining algorithms needs a goal-oriented preprocessing of the data. In practical applications the preprocessing task is very time consuming and has an important influence on the quality of the generated models. In this paper we describe a new approach for data preprocessing. Combining database technology with classical data mining systems using an OLAP engine as interface we outline an architecture for OLAP-based preprocessing that enables interactive and iterative processing of data. This high level of interaction between human and database system enables efficient understanding and preparing of data for building scalable data mining applications. Our case study taken from the data-intensive telecommunication domain applies the proposed methodology for deriving user communication profiles. These user profiles are given as input to data mining algorithms for clustering customers with similar behavior.

1 Introduction

The telecommunication industry is faced right now especially in Germany with a growing competition. "Knowing the customers" is one of the most important steps towards customer-specific pricing and offering special tariffs. Data Mining is a very promising technology to tackle the Customer Relationship Management (CRM) task that becomes more and more important in all industries.

The telecommunications domain is very data-intensive. The immense amount of data makes classical desktop data mining systems hardly applicable. Especially the tasks of data integration and preprocessing are hard to handle. The preprocessing task that is very time consuming in real data mining projects gets more and more difficult.

Recent work emphasize the promising direction of integrating database technology and data mining (see e.g. [7], [9]). We propose merging the scalable database technology using a data mart with classical data mining systems and algorithms. In our approach the interface for this combination consists of an OLAP (online analytical processing) engine that allows a high level of interaction between human and database system. As far as possible the most preprocessing operations are processed in the database system. Additionally, the consistent way of performing preprocessing in a database is scalable and can easily be revoked and reused.

The paper is organized as follows. In section 2 we draft our analysis scenario. The analysis scenario describes our real-world application of handling the complex, high

Y. Kambayashi, M. Mohania, and A M. Tjoa (Eds.): DaWaK 2000, LNCS 1874, pp. 258–264, 2000.
© Springer-Verlag Berlin Heidelberg 2000

dimensional and enormous amount of communication data. We further explain in section 3 the general core of our approach called OLAP-based preprocessing. Taken from the telecommunication domain we present a case study using OLAP-based preprocessing. In section 4 we outline how communication profiles are generated using call detail records and show how these profiles are used for customer segmentation. Section 5 gives an short overview over similar works and section 6 concludes with an outlook on further work.

2 A Telecommunication Analysis Scenario

Telecommunication companies make their business by selling communication minutes to their customers. Considering these minutes as the main product different properties must be distinguished that determine the variation of the product. These properties or dimensions include, for example, the daytime, weekday or weekend, the calling distance etc. Calculating the different possible combinations of attribute values for just these three dimensions gives a rather large number. For example, taking every hour of a day as value for the dimension daytime, distinguishing just between weekday and weekend and considering 10 different calling distances (e.g. city, international, ...) gives 24 x 2 x 10 = 480 different combinations called communication features. The Deutsche Telekom distinguishes internally even many more than just 10 calling distances.

From a customer care point of view the Deutsche Telekom aims at identifying groups of customers which typically buy the same or similar products. Understanding the needs of the customers special offers can be made and the tariffs can be adjusted to the customer behaviour. Realizing this difficult task using data mining we build in a first step a communication profile for each customer. A communication profile should represent the typical calling behavior of a customer over a rather long period of time. Having analyzed the domain and the data given in form of call detail records two difficulties arose. First, a customer buys only one product at a time which is contrary to the classical market basket analysis. Second, the large number of single transactions (call detail records) must be aggregated to a more meaningful level.

In our scenario call detail records of around 4500 private customers have been stored in a panel for 5 years. In the average a private customer has between 3 and 7 phone calls a day. Thus, approximately 12.000.000 call detail records were produced and stored in flat files. In addition to this communication data social-demographic data for these private customers was collected.

After having integrated the communication as well as the social-demographic data in one data mart we started to load the data from the data mart into a data mining system. First, we intended to perform all necessary preprocessing within the data mining system. This approach, however had the heavy drawback that all time-consuming preprocessing steps had to be executed again and again using new data. Additionally, this way of preprocessing within the data mining tool was not scalable, reusable and flexible as necessary for our complex and data-intensive domain.

3 Our architecture for OLAP-based preprocessing

Practical experiences in the development of data mining applications in general have shown, that the sub-phases data connection and integration, data understanding and

preprocessing take the majority of the whole process time. Whereas preprocessing plays a minor role in scientific research it is of highest relevance in real-world applications. The pure application of data mining algorithms in the development process requires only a petty length of time. However, the quality of the generated models is heavily depending on the algorithm specific data preprocessing.

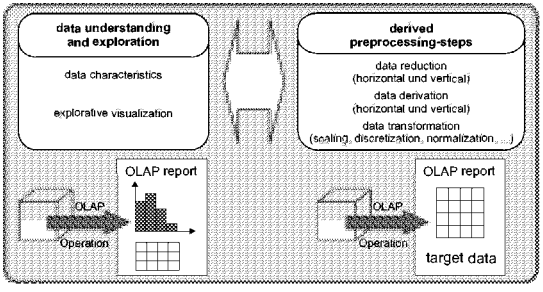


Figure1. Procedure of preprocessing data, in the lower part extend with OLAP functionality

In the upper part of figure 1 the typical procedure of preprocessing the data is shown. Preprocessing is performed in interaction with understanding the data. To understand the data different methods can be applied: One way is to perform multivariate statistical methods to calculate some data characteristics. An other way is the exploratory visualization of the data. After having gained some ideas and hints about the data goal-oriented preprocessing operations can be performed. We distinguish between three groups of preprocessing operations: Data reduction, data derivation and data transformation. The process of data reduction is partitioned into horizontal and vertical reduction. Likewise, data derivation can also be performed in a horizontal and vertical manner. Horizontal derivation adds new individuals to the data set (also called balancing) and vertical derivation means the generation of new attributes by combining existing ones. Data transformation includes operations like applying mathematical functions, discretization or categorization, normalization and replacement of values.

Handling very large amounts of data, the iterative and interactive task of data understanding and preprocessing becomes more and more time consuming. In the following we will show how the preprocessing task described earlier can be performed more efficiently on large amounts of data. The lower part of figure 1 shows the idea of our OLAP-preprocessing framework. We propose using an OLAP-engine to have a flexible way for understanding and preprocessing large datasets. Generating reports with tables and graphs by applying OLAP operations (see lower left part of figure 1) is usually used to obtain valuable information for business decisions. In our case we do not only want to apply OLAP for data understanding or visualization, but also to create a target data set that can be used to generate new hypotheses by applying a data mining algorithm.

The idea described above was realized by the architecture depicted in figure 2. We consider the legacy data to be integrated in a data warehouse or data mart as a necessary prerequisite for efficient data analysis. Therefore, the data integration process is performed only once in our framework and all continuing steps are performed on the data warehouse or data mart. On the one hand side a hyper-cube can be modeled on top of the data mart, on the other hand a knowledge discovery process can be started based on the data in the data mart. An OLAP-engine is used in the modeled hyper-cube.

OLAP is typically performed for the user-driven validation of hypotheses. OLAP functionalities include drill-down, roll-up, slice and dice, pivoting operations for flexible handling and transforming the data (see [1]). The knowledge discovery process consists of various phases that have to be passed (cf. CRISP-DM standard process model for knowledge discovery [3]). The phases typically enclose the tasks of data selection, preprocessing, model generation and interpretation like it is shown in figure 2. Data mining algorithms are executed on the target data set to generate models which have to be interpreted concerning the business relevant questions.

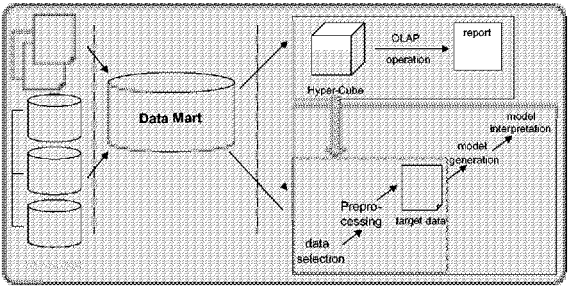


Figure2. Architecture for OLAP-based preprocessing

A target data is derived from applying one of the preprocessing operations described earlier in this section. As indicated in figure 2 by the arrow from the hyper cube to the data preparation part of the knowledge discovery process the data created using the OLAP functionalities on top of the hyper-cube can also serve as preprocessed target data for data mining algorithms. The OLAP-preprocessed data is loaded into the data mining system where further preprocessing can be performed if necessary.

4 A case study

As mentioned in section 2 the Deutsche Telekom uses a panel to analyze the communication behavior of its customers. For 5 years every phone call of around 4500 private customers has been logged with their aggreement. The information stored in anonymous form includes the number of calls per day, the duration of each call, the calling distance, the daytime, weekday etc. Additionally, a well-known market research institute was charged with inquiries at these private customers to get social-demographic information in order to better classify and describe different customer groups. In the present time of a very competitive and unsteady telecommunication market in Germany the Deutsche Telekom tries to extract valuable information from these data sources for goal-oriented marketing actions and new pricing activities. To solve the desired requirements for a quick and efficient analysis of the different information sources it was decided to develop a data mart called PAS (for **p**anel **a**nalysis **s**ystem) that contains all relevant information. Data mining methods should be applied to the PAS such that the data mart serves as a decision support system for marketing and pricing actions.

In a case study using the PAS data mart it was the goal to describe each panel customer by a communication profile and build groups of customers that have similar profiles. A communication profile consists of typical characteristics (e.g. a similar number of international calls at certain times on a weekday) of the customer. Generating the

desired communication profiles was realized using the idea of OLAP-preprocessing that was introduced in section 3.

4.1 Application of OLAP-Preprocessing

Our approach presented in section 3 was applied to the case study described in the last section. To derive the target data we performed the most preprocessing steps by applying various OLAP functions. Because of the complexity indicated in section 2 we restricted the communication data to 3 month for building a customer profile.

In a first approach, the calling minutes of each customer were aggregated over the hour in which the call started, every day of the week and the calling distance. However, this approach was cancelled since the number of communication features (over 1000!) was too complex to handle. An exploratory analysis revealed that the data was too detailed. In a second approach, the calling minutes were summed up over 6 hour slots starting from 0am to 6am, 6am to 12pm and so on. Furthermore, we distinguished just between weekdays (Monday thru Friday) and weekend (Saturday and Sunday). The values for the dimension calling distance was also cut to three. As a result we got the number of 24 communication features (4 x 2 x 3 = 24) used to describe the communication behavior of a single customer. On the left side of figure 3 an average of all panel customers with the chosen 24 communication features is depicted. The middle of the x-axis separates weekdays from weekends, while the first 12 communication features represent the weekdays section. Weekdays are further separated in three areas depending on the calling distance (city, regional, national). Furthermore every calling distance is subdivided into the 4 time windows mentioned earlier. The y-axis represents the average communication minutes in dependence of the communication feature.

The desired aggregated information was returned by an OLAP tool in form of a single table that contained all communication transactions of the panel customers within 3 months. In a next OLAP preprocessing step the "pivot" functionality transformed the column representation of a single customer to the more appropriate form of a row representation. A single row consists of a customer id and the calling minutes aggregated for each of the 24 communication features. Using simple visualization techniques to view the distribution of values for each communication feature it became evident that all communication features are more or less left-slanted distributed. As a prerequisite for applying the cluster algorithm the data had to be transformed from a left-slanted distribution to a symmetric distribution. This was achieved by transforming the calling minutes of each communication feature by the logarithmic function.

4.2 Clustering and Interpretation

Using the well-prepared and preprocessed data the task of building various customer segments with its characteristic communication behavior could be addressed. The well-known and successfully approved k-means clustering method (see [6]) was selected and applied to the data. The number of clusters k was set to 10. To determine the number k, we executed a hierarchical clustering algorithm on a sample set (cf. [8]). The absolute number of elements in the 10 clusters varies from 109 to 777 private customers. Analyzing the smallest cluster revealed that these private customers were identified as outliers. Their calling minutes were significant above the average communication behavior of

private customers and look more like small enterprises. The remaining nine clusters yielded a concentration of private customers with rather homogeneous communication behavior within each cluster. Indeed, the found clusters built a very good basis for segmenting different groups of private customers. The average profile of each cluster (see left side of figure 3) could be visualized by using so called error diagrams. For each of the 24 communication features the mean value and a 95-% confidence interval is calculated and represented for each cluster. By using this simple visualization technique very interesting differences between the clusters could be detected concerning the usage of special communication features.

On the right hand side of figure 3 a cluster with 777 members is represented. The communication behavior of the cluster members significantly differs from the average communication behavior of all 4500 panel customers. This cluster is very similar in city calls on weekdays and weekends compared to the average of all customers. However, all other communication features are obviously different (see figure 3).

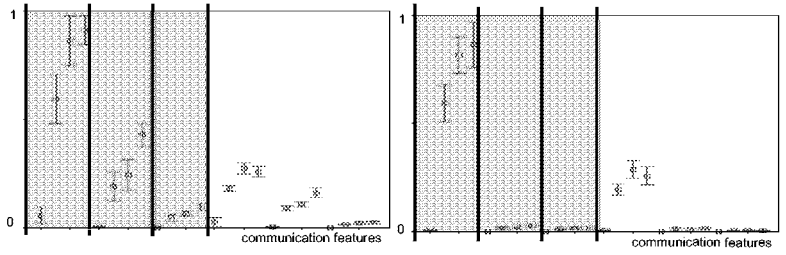


Figure3. Communication profiles on left side of all customers, on right side of one cluster

As mentioned earlier in this section the PAS data mart contains additional social-demographic data derived from the customer inquiries. These include, for example, information concerning the size of the household or the net income of the customer. The demographic data can be accessed by an OLAP tool like the communication data and is added in form of columns to the table described in section 4.1 such that it can be used for further analysis. It is intended to use this information for generating intensional rules that explain better each derived cluster.

5 Related Work

Few methodologies have been proposed for the preprocessing task. In [4] it is described how recommendations for preprocessing steps can be derived for classification tasks that base on the calculation of data characteristics. Calculating data characteristics on complex and very large data sets is very difficult and in some cases impractical and rely on statistical assumptions that do not hold in real world domains.

The relationship between OLAP and data mining is examined by Parsaye (see [9]). He describes an architecture that combines OLAP and data mining applications and shows how data mining applications depend on different aggregation levels. Han's group (see [7]) focuses its research on the area of OLAP mining. One of the motivations for OLAP mining was the perception that different aggregation levels were necessary for their pattern analysis. Both approaches concentrate on data mining algorithms but hardly on other parts of the knowledge discovery process.

6 Conclusion & Further Work

In this paper we presented an approach for efficient and scalable data understanding and preprocessing in real world domains. Using OLAP technologies for preprocessing offers quite a lot of advantages: OLAP applications are usually interactive and can handle very well large data sets.

Our approach presented in this paper aims at simplifying the problem of preprocessing real world data based on an existing OLAP environment. The performed preprocessing steps within the OLAP environment leads to well prepared target data for the following clustering task. That way we were able to derive interesting clusters. The communication profiles derived in the case study are based on the assumption that 24 communication features are suited to describe the customer behavior. Existing works that build user profiles (e.g. [2], [5]) in a similar way make pretty much the same assumptions, but use different aggregation levels. Applying our architecture for OLAP-based preprocessing we are able to reconstruct different aggregation levels in a very efficient and easy way. The determination of the right aggregation level is a nontrivial task. In our further research we intend to investigate the influence of different aggregation levels on clustering algorithms and on other data mining methods.

Acknowledgements The work presented in this paper was partially financed by an internship of the Deutsche Telekom AG.

References

1. S. Chaudhuri and U. Dayal: An Overview of Data Warchousing and OLAP Technology. SIGMOD Record, Volume 26, p. 65-74, 1997.
2. Qiming Chen, Umesh Dayal, Meichun Hsu: OLAP-based Scalable Profiling of Customer Behavior. In Proceedings of the First International Conference on Data Warehousing and Knowledge Discovery, (DaWaK '99), 1999.
3. P. Chapman, J. Clinton, T. Khabaza, T. Reinartz, R. Wirth: The CRISP-DM Process Model, (http://www.crisp-dm.org), 1999.
4. R. Engels and C. Theusinger: Using a Data Metric for Offering Preprocessing Advice in Data Mining Applications. Proceedings of the 13th European Conference on Artificial Intelligence (ECAI 98), Springer, Brighton, p. 430-434, 1998.
5. T. Fawcett and F. Provost: Combining Data Mining and Machine Learning for Effective User Profiling. In Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96), AAAI Press, 1996.
6. K. Fukunaga: Introduction to Statistical Pattern Recognition. San Diego, CA, Academic Press, 1990.
7. J. Han: OLAP Mining: An Integration of OLAP with Data Mining. Conference on Data Semantics (DS-7), 1997.
8. L. Kaufman and P.J. Rousseeuw: Finding Groups in Data: An Introduction to Cluster Analysis. New York, Wiley, 1990.
9. K. Parsaye: OLAP and Data Mining: Bridging the Gap. Database Programming and Design, Volume 10, p. 30-37, 1998.

Meta-queries - Computation and Evaluation

Rachel Ben-Eliyahu-Zohary[1] and Ehud Gudes[2]

[1] Dept. of Communication Systems Engineering
Ben-Gurion University of the Negev
Beer-Sheva 84105, Israel
rachel@bgumail.bgu.ac.il
[2] Dept. of Computer Science
Ben-Gurion University of the Negev
Beer-Sheva 84105, Israel
ehud@cs.bgu.ac.il

Abstract. *Metaquery* (also known as *metapattern*) is a datamining tool useful for learning rules involving more than one relation in the database. A metaquery is a template, or a second-order proposition in a language that describes the type of pattern to be discovered. In an earlier paper we discussed the efficient computation of *support* for Meta-queries. In this paper we extend this work by comparing several support computation techniques. We also give real-life examples of meaningful rules which were derived by our method, and discuss briefly the software environment in which the meta-queries were run (the FLEXIMINE system). Finally we compare Meta-queries to Association rules and discuss their differences.

1 Introduction

With the tremendous growth in information around us, datamining is emerging as a vital research area among the AI and Databases communities [6]. Metaquerying [8, 9] is a very promising approach for datamining in relational or deductive databases. Metaqueries serve as a generic description of the class of patterns to be discovered and help guide the process of data analysis and pattern generation. Unlike many other discovery systems, patterns discovered using metaqueries can link information from many tables in databases. These patterns are all relational, while most machine-learning systems can only learn propositional patterns.

Metaqueries can be specified by human experts or alternatively, they can be automatically generated from the database schema. Either way, they serve as a very important interface between human "discoverers" and the discovery system.

A metaquery R has the form

$$T \longleftarrow L_1, ..., L_m \tag{1}$$

where T and L_i are literal schemas. Each literal schema L_i has the form $Q_i(Y_1, ..., Y_{n_i})$ where all non-predicate variables Y_k are implicitly universally quantified. The expression $Q_i(Y_1, ..., Y_{n_i})$ is called a *relation pattern*. The right-hand-side $L_1, ..., L_m$ is called the *body* of the metaquery, and T is called the *head* of the metaquery.

Y. Kambayashi, M. Mohania, and A M. Tjoa (Eds.): DaWaK 2000, LNCS 1874, pp. 265–275, 2000.
© Springer-Verlag Berlin Heidelberg 2000

The predicate variable Q_i can be instantiated only to a predicate symbol of the specified arity n_i. (note, however, that in our real-life experiments we allowed the projection of relation with larger arity into such predicate). The instantiation must be done in a way which is consistent with the variable names.

student	course
Ron	Calculus1
Dana	Algebra
Ana	Intro to CS

course	department
Calculus1	mathematics
Algebra	mathematics
Intro to CS	computer science

student	department
Ron	mathematics
Dana	mathematics
Ana	computer science

Fig. 1. The relations student-course, course-department, and student-department

For example, suppose that you have a database having the relations depicted in Figure 1. Let P,Q, and R be variables for predicates, then the metaquery

$$R(X, Z) \longleftarrow P(X, Y), Q(Y, Z) \tag{2}$$

specifies that the patterns to be discovered are transitivity relations
r(X,Z) $\longleftarrow$ *p(X,Y), q(Y,Z)* where p,q, and r are specific predicates. One possible result of this metaquery on the database in Figure 1 is the pattern

$$\text{stud-dep}(X, Z) \longleftarrow \text{stud-course}(X, Y), \text{course-dep}(Y, Z) \tag{3}$$

which means intuitively that if a student takes a course from a certain department then he must be a student of that department. Note that some or all of the attributes may be instantiated, e.g. the rule:
stud-dep(X,math) $\longleftarrow$ *stud-course(X,Y), course-dep(Y,math)* means that the above statement is true for math students.

It turns out that the notion of Meta-query is related to the concept of Association rules [1]. It is also related to Schema query languages such as *SchemaSQL*. The discussion section will discuss these relationships.

1.1 The notion of *support* and *confidence* for metaqueries

Suppose that we are given the metaquery (2) again, but instead of the relations shown in Figure 1, we have added the corresponding rows and columns shown in Figure 2 (the relation course-department is the same as in Figure 1).

student	course
David	Intro to CS

student	department	sex
Ron	mathematics	M
Dana	mathematics	F
Ana	computer science	F
David	mathematics	M

Fig. 2. The new relations student-course and student-department

The rule (3) doesn't hold in all cases anymore. But we can say that it holds in 75% of the cases, or in other words, that it has *confidence* 0.75.

On the other hand, let us consider the Meta-query:

$$R(X, Y, W) \longleftarrow P(X, Z), Q(Y, Z) \tag{4}$$

One of its instatiated rules may be:

$$\text{stud-dep}(X, Z, sex =' F') \longleftarrow \text{stud-course}(X, Y), \text{course-dep}(Y, Z) \tag{5}$$

and this rule has a confidence of 100%! even though it involves only one half of one of the relations. This motivates of-course the notion of *support* similar to that notion in association rules [1].

Hence, each answer to a metaquery is a rule accompanied by two numbers: the *support* and the *confidence*. The threshold for the support and confidence is provided by the user. Intuitively, the *support* indicates how frequently the body of the rule is satisfied, and the *confidence* indicates what fraction of the tuples which satisfy the body also satisfy the head. Similar to the case of association rules, the notions of *support* and *confidence* have two purposes: to avoid presenting negligible information to the user and to cut off the search space by early detection of low support and confidence. Formally, given a rule

$$t \longleftarrow r_1, ..., r_m, \tag{6}$$

let J denote the relation which is the equijoin of $r_1, ..., r_m$, and let Jt be the relation which is the equijoin of J and t. Where y and x are some relations, let y_x, be the projection of y over the fields which are common to y and x, and let $|x|$ be the number of tuples in x. For each i, $i = 1...m$ define S_i to be the fraction $\frac{|J_{r_i}|}{|r_i|}$. In [2] we first presented the following definition for support of meta-queries: The *support* of the rule (6) is the maximum over S_i, $i = 1...m$. Less formally, the support is the maximum fraction of any relation r_i in J. The *confidence* of (6) is the fraction of t that appears in J, or, formally, the *confidence* of (6) is $\frac{|Jt_J|}{|J|}$. The support that we have defined has the following useful property:

Claim. For any two relations r_i and r_j in the rule (6) that have at least one common attribute variable, S_i is not bigger the the fraction of r_i that participates in the equijoin between r_i and r_j.

This property enables us to get an upper bound on the support of the rule (6) by performing pairwise equijoins instead of equijoin of all the relations in the body of the rule. In [2] we compared our definition of support to that of Shen et al. [9], and showed that it is more general and intuitive.

The main part of [2] was devoted to present efficient algorithms to compute support. We briefly review these algorithms here in Section 2. In Section 3 we compare the performance of two of the Support computing algorithms: *Brave* and *Cautious*. In Section 4 we present some examples of real-life rules derived by our Meta-query method and discuss its software environment (the Fleximine system [5]). In Section 5 we compare Meta-queries to association rules and schema query languages, and we conclude in Section 6.

2 Support computation algorithms

How hard is answering a metaquery? In [2] this question was shown to be NP-hard in the number of attributes and relations. The naive algorithm is to try to instantiate each predicate and variable with all possible relations and attributes subject to the unification constraints, for each such instantiation to compute the support, and for those instantiations which pass the support threshold to compute the confidence. In practice though, many such instantiations will have very low support and detecting this low support early, will make the above computation much more efficient! this is discussed next.

The process of answering a metaquery can be divided into two stages. In the first stage, which we call the *instantiation stage*, we are looking for sets of relations that match the pattern determined by the metaquery. In the second stage, which we call the *filtration stage*, we filter out all rules that match the pattern of the metaquery but do not have enough support and confidence.

The process of instantiating a metaquery is similar to solving a Constraint Satisfaction problem (CSP) [4] where we are basically looking for all solutions of the CSP problem. In our experiments we used a very simple CSP algorithm (forward checking with Back-jumping [3]) but other more advanced algorithms may be used. The basic instantiation algorithm was presented in [2] and will not be shown here.

The filtration stage itself is composed of two steps: filtering out rules with low support, and filtering out rules with low confidence. We compute confidence only for rules with sufficient support. In our research we have focused on the following algorithms for computing support:

The Join approach the straightforward way: computing the equijoin of the body of the rule, then computing S_i (S_i as defined in section 1.1) for each relation in the body, and then taking the maximum.

compute-support$(r_1, ..., r_m)$
Input: Set of relations $r_1, ..., r_m$, where each of the attributes is bound to a variable. A support threshold $MinSupport$.
Output: if the rule who's body is $r_1, ..., r_m$ has support equal or larger than $MinSupport$, return the support, otherwise return -1.
1. $RelSetCopy = RelSet = \{r_1, ..., r_m\}$; $LowSupp =$ **true**;
2. While ($RelSet \neq \emptyset$) **and** $LowSupp$ do
  1. Let $r \in RelSet$;
  2. $s = S_i$-upbound$(r, RelSetCopy)$;
  3. **if** $s \geq MinSupport$ then $LowSupp =$ **false**
     else $RelSet = RelSet - \{r\}$;
3. **If** $LowSupp$ then return -1
  else return Join-support$(r_1, ..., r_m)$

Fig. 3. computing support for a rule body

The histogram approach Using histograms for estimating support. Computing the exact support only for rules with high estimated support.

The histogram + memory approach same as the histogram approach, except that we store intermediate results in memory, and reuse them when we are called to make the same computation.

The procedure Join-support$(r_1, ..., r_m)$ computes the support S_i of each relation by performing the Join as explained in section 1.1. and then returns $\text{Max}\{S_i | 1 \leq i \leq m\}$. The other two approaches compute an upper bound on the support and then compute the exact support only for rules with high enough upper bound of support. The idea is summarized in Algorithm compute-support in Figure 3. Note that once one relation with high S_i is found, we turn to compute the exact support using Join.

The procedure S_i-upbound called by the algorithm compute-support returns an upper bound for the value S_i for a single relation r_i in the body of the rule. This can be done by one of the two procedure: S_i-upbound-brave or S_i-upbound-cautious, shown in Figures 4 and 5, respectively. The basic idea is that an upper bound can be achieved by taking the join of a relation r_i with any other relation with which r_i has a variables in common. Procedure S_i-upbound-brave does this by picking one arbitrary relation with which r_i has a common variable, and procedure S_i-upbound-cautious does this by considering *all* relations with which r_i has variables in common, and taking the minimum. Procedure S_i-upbound-cautious works harder than procedure S_i-upbound-brave but it achieves a tighter upper bound and hence can save more Join computations.

The procedure *upbound* called by procedure S_i-upbound-cautious or procedure S_i-upbound-brave uses either the the histogram approach or the his-

togram+memory approach. The histograms approach exploits the fact that histograms are easy to construct and are quite useful for support estimation. The algorithms to compute the support using histograms (i.e. the procedure up_bound_histo) were also presented in [2] and will not be repeated here.

The evaluation of the above support computation algorithms was also reported in [2]. The three strategies described above were compared and tested using only the procedure S_i-upbound-brave. The main conclusion as expected was that the two estimating methods were better than the Join method especially when the support threshold was large. This, since in those cases, they had a better oppurtunity to cut the search space.

S_i-**upboud-brave**(r_i,S)
input: A relation r_i and a set of relations S.
output: An upper bound on S_i for a rule whose body is $S \bigcup \{r_i\}$.
1. $s = 1.0$
2. If there is $r' \in S$ such that r' and r_i have a common variable X then s =upbound$(r_i, r', att(r_i, X), att(r', X))$;
3. return s;

Fig. 4. computing S_i bravely

S_i-**upboud-cautious**(r_i,S)
input: A relation r_i and a set of relations S.
output: An upper bound on S_i for a rule whose body is $S \bigcup \{r_i\}$.
1. $s = 1.0$
2. **for** each relation r' in S such that r_i and r' have variables in common
do **for** each common variable X of r_i and r'
do
s' =upbound$(r_i, r', att(r_i, X), att(r', X))$;
if $s' < s$ then s=s';
3. return s;

Fig. 5. computing S_i cautiously

2.1 Comparing the Brave vs. Cautious

As was mentioned above, in [2] we only used the *Brave* algorithm for computing support. We later extended the experiment and measured the performance of Brave vs. Cautious.

In Figure 6 we see the performance of the Brave vs. the Cautious algorithms for two support threshholds. The concept of "Miss" means that the upper-bound computed by the algorithm was above the threshold while the actual support value was below the threshold. As can be seen, from the graphs, the total number of misses for the first 20 queries is more for the Brave then for the Cautious and the difference is reduced as the support threshold is increased. The explanation may be that for low threshold values the mistakes made by the algorithms are more sensitive. This behaviour is not consistent though with very high support threshold, so it needs further investigation. In anycase, it seems that if the case is that many instant queries will "fail" because of low support (not the case reported here), the difference between the algorithms becomes less significant, and the Brave method should be satisfactory.

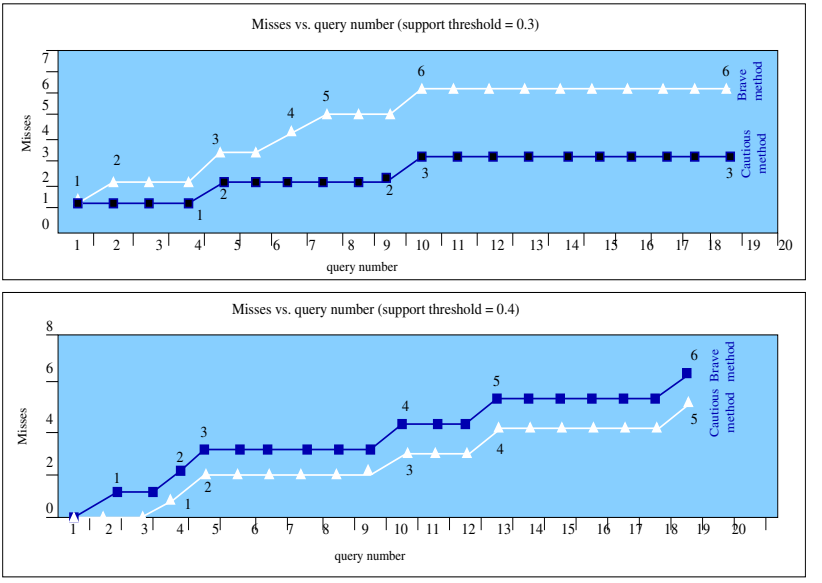


Fig. 6. Comparison of Brave vs. Cautious

3 Meta-queries implementation in FlexiMine

FlexiMine [5] is a prototype KDD system currently being developed at Ben-Gurion University for testing techniques and algorithms for Data Mining and Knowledge Discovery and their evaluation in the context of real-life databases and users. The emphasis of this system is on integration of most KDD operations, (including database access and selection, preprocessing and abstraction, data-mining algorithms such as: Association rules, Baysian knowledge-bases, Decision-trees), and on **extensibility**. Thus, the system facilitates incorporation of new algorithms or their improved variants, and convenient extension of support to new databases or abstraction hierarchies.

Usually a user of FlexiMine is presented with a screen containing the schema of the selected relational database. The user then goes through a process of defining the specific set of attributes on which she wants to perform data-mining algorithms. The results of this selection is a **flat** file of records, since algorithms such as Assoc. rules or Decision-trees work on flat files. As part of this process, the user may apply the process of **Abstraction**. In real-life databases there may be attributes with continous numeric values. These attributes need to be abstracted into large grain categories, in order for the particular mining algorithm to succeed finding item with high enough support. This process was described in detail in [5].

The Meta-queries algorithm is different of other data-mining algorithms in that it access the original database directly without going through a selection procees. This is obviously needed since Meta-queries use heavily the Schema information. On the otherhand, the Abstraction process is important! There are two ways of handling abstractions. Some fields have their default abstractions which are used by the Meta-queries. Other fields may be instantiated during the process and may not have any linked abstraction. Such fields are abstracted during the instantiation process using some huristics which takes into account the number of distinct values in the domain.

The metaquery module is implemented in SQL embedded in C++. At present, we use syntax similar to Prolog or Datalog. Since, in our system, it is possible to replace one of the relation or field variables with a constant (in order to specify a pattern involving a specific field or a specific relation), such a constant is enclosed in quotes. For example, in the following Meta-query:

"Course_Student"(X,Y,"Grade"),"Students"(X,W,"Age") ->A (Y,Z,U)

we are looking for rules connecting grade results and students age to the name of the course (e.g. a possible rule is: young students are likely to get 80 or higher in Intro to CS...)

We tested the Meta-queriy algorithm on several databases. One of these databases was an Hospitalization database of about 600,000 records. Following are some interesting instants of rules we received from this database. The Meta-query was as follows:

"diagnosis"(X,"name"), "hospital_record(X, Y,) ->Z(X, U,V)

which means: Find relationships between diagnosis and name of disease, the hospital record, and any other relation in the database on the right side. We

also used the following abstraction on Age which is a discrete numeric value: 'adult' means age between 26-54. 'old' - 55 - 95, and 'young' - 1-25

1. *diagnosis(patient_id,name ='Psychiatric_disorders'),*
   hospital_record(patient_id,a,reason='birth')
   ->personal(patient_id,age_group='adult',sex='female')
   Support=0.025 Confidence=0.790
2. *diagnosis(patient_id,name='Neurologic_disease')*
   hospital_record(patient_id,a,release_status = 'death')
   ->personal(patient_id,age_group='old',sex='male')
   Support=0.002 Confidence=0.344
3. *diagnosis(patient_id,name='Neurologic_disease')*
   hospital_record(patient_id,a,release_status = 'death')
   ->personal(patient_id,age_group='old',sex='female')
   Support=0.002 Confidence=0.281
4. *diagnosis(patient_id,name='Immunologic_desease')*
   hospital_record(patient_id,a,release_status = 'healthy')
   ->personal(patient_id,age_group='young',marital_status='single')
   Support=0.004 Confidence=0.302

The first instance is quite obvious and often occurs in Association rules. 79% of the patients diagnosed with psyciatric-disorder and were admitted because they gave birth were women of the age-group 'adult' The second and third instances show a slight difference between females and males for death from the same (hospitalization) reason. The fourth instance shows other attributes that were instantiated by the Meta-query algorithm.

Note also that the support values for the above queries is quite low. If we would have abstracted the Diagnosis disease codes into higher level diseases, we probably would have gotten higher values of support. This shows the importance of Abstraction in the Meta-query system, and we are currently implementing such abstraction based on the ICD9 clinical coding scheme.

4 Discussion

4.1 Association rules vs. Meta-queries

From the previous discussion and examples it seems that there is a large similarity between Meta-queries and Association rules. Yet, although in principle the same rules or knowledge may be derived by both (or by Decision trees and other methods), there are still several significant differences.

The first issue is the use of **schema** information. Rules from Meta-queries include the Relation names in the rule. Now, in most cases these names have important meaning which is directly observable by looking at the rule, while in case of Association rules, one has to remember where the various items came from and what is their meaning. Furthermore, if the database contains many views which represent particular "slices" of the database, one can find interesting queries which involve such views (with high support).

A second issue is that Meta-queries operate on the database itself, while association rules require a phase of Selection and Pre-processing. The result of this phase must be a well organized flat file with well defined transactions. This phase, usually ignored by most researchers, is quite painful and is not at all obvious. The main problem here is the problem of **Normalization**. Assume a common database of students and courses with three relations:

Students(Stud_Id, Name,...), Course(Course_Id,Name,...)
Student_Course(Course_Id, Student_Id, Grade,...)

If one joins the relations Students and Student_Course and if one looks for rules involving attributes such as Age or Sex and Grade, one will get wrong results, since Age and Sex values will be biased towards students taking many courses! The solution is of-course Denormalization and the construction of *Transactions* where each transaction corresponds to one student, and course grades are reported as individual new fields (or their maximum, or average). This process is supported directly in Fleximine and is called there **Aliasing** [5]. This process though is not automatic, the user must be aware of it, and must direct the system what type of Denormalization or Aliasing to perform. Such a process is unnecessary with Meta-queries, since the Join structure is preserved.

Finally, meta-queries can discover more "general" knowledge than Association rules. Consider the Meta-query (2) in Section 1, and the instance (3). Such a rule cannot be discovered directly by an Association-rule algorithm, because it involves a constraint on set of values. (in some sense, the resulted rule is an **Integrity constraint**). One can define during the pre-processing stage a new attribute which will get the value of 1, each time the join row of a student and the course it takes results with equality of the student and course departments. But, in order to do that, one has to know the desired result during the pre-processing stage, clearly an unrealistic assumption! Such a pre-knowledge is not required at all by the Meta-query method. For all these reasons, meta-queries are useful along with Association rules for Knowledge discovery.

4.2 Comparison with Schema query languages

Recently, there is a development of query languages over the *schema* as well as over the content of a database. We briefly compare Meta-queries to one such language, *SchemaSQL* [7]. The motivation for such languages is the querying of Web data, and the integration of hetrogeneous databases. One must be able to query the schema in those cases. The similarity, with Meta-queries is that in both, variables can "run" on relations and attributes names. There are, however, significant differences because of the different motivation. First, the result of a SchemaSQL query is a single reation (or set of relations), while the result of a Meta-query is a set of instantiated queries, each is then evaluated separately against the original database! Second, the notions of support and confidence are essential for Meta-queries and help to cut the search space, as well as to give only relevant results. These concepts are completely missing from SchemSQL queries. Thirdly, some of the operators of SchemaSQL (e.g. split) are non-relevant to Meta-queries, since they may create new relations from tuples (values) with

very low support. On the other-hand other operators (e.g. unfold) may be useful to solve the Denormalization problem mentioned above for Association rules. Finally, some of the optimization techniques used by schemaSQL may be useful for Meta-queries too, but this requires further investigation.

5 Conclusion

This paper extends previous research on metaqueries in several ways. We first review our notion of support for a rule generated according to a pattern, and we present efficient algorithms for computing support. We then compare the performance of two algorithms for computing support bounds: the Brave and Cautious algorithms. Then we discuss the implementation of Meta-queries withing the FlexiMine system framework. We also present some results of rules generated from meta-queries on a real-life database. Finally, we compare Meta-queries to Association rules and to Schema query languages and discuss their corresponding advantages. Future work will both extend the power and syntax of the Meta-query system, and will investigate their optimization and performance under various conditions.

References

1. Agrawal, S., Srikant, R.: Fast algorithms for mining association rules. Proceedings of the 20th VLDB Conference, Santiago, Chile, (1994).
2. Ben-Eliahu, R., Gudes, E.: Towards efficient meta-querying. Proceedings of IJCAI-99, Stockholm, Sweden, (1999).
3. Dechter, R.: Enhancement schemes for constraint processing: Backjumping, learning, and cutset decomposition. The AI journal, **41** (1990) 273–312.
4. Dechter, R.: Constraint networks. Encyclopedia of Artificial Intelligence, John Wiley, (1992) 276–285.
5. Domshlak, C., Gershkovich, D., Gudes, E. Liusternik, N., Meisels, A., Rosen, T., Shimony, S.: FlexiMine - A Flexible Platform for KDD Research and Application Construction. Technical Report FC-98-04, Ben-Gurion University, (1998), Shorter version in KDD-98, Fourth International Conference on Knowledge Discovery in Databases, 1998.
6. Fayyad, U., Piatetsky-Shapiro, G., Smyth, P., Uthurusamy. R.,: Advances in Knowledge Discovery and Data Mining. AAAI Press/MIT Press, (1996).
7. Lakashmanan, L.V., Sadri, F., Subramanian, S.: n efficiently implementing SchemaSQL on a SQL database system. Proceedings of 25th VLDB conference, Edinburgh, Scotland, (1999) 471-482.
8. Shen, W., Leng, B.: A metapattern-based automated discovery loop for integrated data mining- unsupervised learning of relational patterns. IEEE Trans. on Knowledge and Data Engineering **8(6)** (1996) 898–910.
9. Mitbander, B., Shen, W, Ong, K., Zaniolo, C.: Metaqueries for Data Mining. Advances in Knowledge Discovery and Data Mining, AAAI/MIT press, (1996) 375–397

Partitioning Algorithms for the Computation of Average Iceberg Queries

Jinuk Bae and Sukho Lee

School of Computer Science and Engineering, Seoul National University, Korea
oblody@db.snu.ac.kr, shlee@cse.snu.ac.kr

Abstract. Iceberg queries are to compute aggregate functions over an attribute (or set of attributes) to find aggregate values above some specified threshold. It's difficult to execute these queries because the number of unique data is greater than the number of counter buckets in memory. However, previous research has the limitation that average functions were out of consideration among aggregate functions. So, in order to compute average iceberg queries efficiently we introduce the theorem to select candidates by means of partitioning, and propose POP algorithm based on it. The characteristics of this algorithm are to partition a relation logically and to postpone partitioning to use memory efficiently until all buckets are occupied with candidates. Experiments show that proposed algorithm is affected by memory size, data order, and the distribution of data set.

1 Introduction

[1] The recent research[1][3] has paid attention to iceberg problem. Iceberg problem in database means the relation between a lot of data and a few results is similar to it between an iceberg and the tip of one. Iceberg CUBE problem was introduced in [3]. This problem is to compute only those group-by partitions with an aggregate value above some minimum support threshold. Iceberg queries were introduced in [1]. These queries have four characteristics; (1) computing aggregate functions (2) about large data (3) of which domain size(the number of unique data) is greater than the number of counter buckets, and (4) returning results above threshold.

By the following cases it's necessary to compute them. One is when the amount of data is very large like data warehousing[1][3], market basket analysis[4][8][9] in data mining, clustering[10][11] and so on. Another is when the sizes of data are large like multimedia database[5][6]. The other is when memory is limited like an embedded system[12]. However, average functions have been out of consideration until now. It's because proposed algorithms in [1] are based on coarse counting principles[7][8] and the computation of average by these principles will generate false-negatives(some results are missed). We'll discuss further into this matter in section 2.

[1] This work was supported by the Brain Korea 21 Project.

Y. Kambayashi, M. Mohania, and A M. Tjoa (Eds.): DaWaK 2000, LNCS 1874, pp. 276–286, 2000.
© Springer-Verlag Berlin Heidelberg 2000

```
SELECT target1, target2, ..., targetn, avg(rest)
FROM R
GROUPBY target1, target2, ..., targetn
HAVING avg(rest) >= T
```

(a) Average Iceberg Query (AIQ)

nationality (target1)	job (target2)	income (rest)
Korea	professor	1000
America	manager	1200
China	doctor	2000
Korea	professor	1500

(b) relation R

Fig. 1. Average Iceberg Query and relation R

In this paper, we'll focus on only average iceberg queries(AIQ). An example of AIQ is shown in Fig. 1., based on a relation R($target_1, target_2, ..., target_n$, $rest$) and a threshold T. In this relation, $rest$ attribute(income) have to be numeric. For the computation of AIQ, we'll introduce the theorem to select candidates by means of partitioning, explain BAP algorithm based on it, and improve BAP into POP algorithm. The idea of partitioning is similar to it proposed in [4] for mining association rules. The difference is that the amount of data in each partition is all the same in [4], but so is the number of unique data in this paper.

The rest of this paper is structured as follows. In section 2 we discuss the limitation in related work. In section 3 and 4 we introduce and prove the theorem to select candidates, and present partitioning algorithms based on this theorem. In section 5 we evaluate our algorithms. We conclude in section 6.

2 Related Work

Iceberg CUBE problem introduced in [3] is to compute only those group-by partitions with an aggregate value above some minimum support threshold. The basic CUBE problem is to compute all of the aggregates as efficiently as possible. The chief difficulty is that the CUBE problem is exponential in the number of dimensions: for d dimensions, 2^d group-bys are computed. So, [3] introduced iceberg CUBE problem and used the pruning technique similar to the pruning in Apriori[9]. For example, if <a3, *> and <*, b2> made minimum support, then <a3, b2> would be added to the candidate set. If <a3, *> met minimum support but <*, b2> didn't, then <a3, b2> can be pruned.

Iceberg queries problem was introduced in [1]. [1] proposed algorithms based on coarse counting. The most important feature is to use hashing-scan in coarse counting. The simplest form of this technique uses an array A[1..m] as counter buckets and a hash function h. In the first scan, each target is hashed and the counter bucket for it is incremented by one. After hashing-scan, targets in buckets above specified threshold become candidates. Next scan is performed about only candidates. This method can be extended to sum functions by incrementing counter not by one, but by value.

However, both the pruning technique proposed in Apriori[9] and coarse counting principle can't be applied to average functions. If this is applied to average

Average Threshold = 3.5

relation

professor	5
manager	1
doctor	4
professor	3

h(professor)=1
h(manager) = 1
h(doctor)=2

counter after coarse counting

hash value	sum	count
1	9	3
2	4	1

Fig. 2. A False Negative Example

functions, a false negative example like Fig. 2 will happen. {professor} meets threshold 3.5 because of sum=8 and count=2, but {professor} is omitted from candidates. It's because average is 3 (sum=9 and count=3) in the hash bucket 1 with {professor} and {manager}. That is, {manager} below threshold prevents {professor} from being included in candidates. This matter was discussed in [2], too.

3 Selection of Candidates

In this section, we define notations and prove the theorem to select candidates.

3.1 Notations

For simplicity, we present our algorithms in the next section in the context of a relation R, with $< target, rest >$ pairs. We assume for now we are executing a simple average iceberg query that groups on the single target in R, as opposed to a set of targets. Our algorithms can be easily extended for multiple target sets.

A partition $p \subseteq R$ refers to any subset of the tuples contained in R. Any two different partitions are non-overlapping, i.e., $p_i \cap p_j = \emptyset, i \neq j$, and union of all partitions is the relation, i.e., $p_1 \cup p_2 \cup ... \cup p_n = R$. Each bucket in counter C consists of $< target, sum_val, cnt_val >$. sum_val is the sum of $rest$ of tuples with the same $target$. cnt_val is the number of tuples with the same $target$. And, |C| is defined as the number of all buckets in counter C.

The form of functions used in this paper is $function_name(target, range)$. When there are a target t, a partition p, and relation R consisted of n partitions, $count(t, p)$ is defined as the number of tuples, of which $target$s are the same as t within p, each function is as follows.

$$sum(t,p) = \sum^{count(t,p)} rest, \quad avg(t,p) = \frac{\sum^{count(t,p)} rest}{count(t,p)}$$

$$count(t,R) = \sum_{i=1}^{n} count(t,p_i)$$

$$sum(t,R) = \sum_{i=1}^{n} sum(t,p_i) = \sum_{i=1}^{n} \sum^{count(t,p_i)} rest$$

$$avg(t,R) = \frac{sum(t,R)}{count(t,R)} = \frac{\sum_{i=1}^{n} \sum^{count(t,p_i)} rest}{\sum_{i=1}^{n} count(t,p_i)}$$

3.2 Theorem for Selecting Candidates

Theorem 1. The average threshold T is given and relation R is partitioned into n partitions$(p_1, p_2, ..., p_n)$. Then, for any target t

$$avg(t,R) \geq T \Rightarrow (\exists i \; avg(t,p_i) \geq T), \quad i = 1,2,..,n.$$

The converse preposition is not satisfied.

Proof. Suppose that there is a target t, of which average value doesn't meet T in each partition but in R. From this assumption, we have

$$\forall i \; avg(t,p_i) = \frac{\sum^{count(t,p_i)} rest}{count(t,p_i)} \prec T, \quad i = 1,2,..,n. \tag{1}$$

Multiplying both sides of (1) by $count(t,p_i)$ yields

$$\forall i \quad \sum^{count(t,p_i)} rest \prec T \cdot count(t,p_i), \quad i = 1,2,..,n. \tag{2}$$

Replacing upper part of $avg(t,R)$ with (2), we obtain

$$avg(t,R) = \frac{\sum_{i=1}^{n} \sum^{count(t,p_i)} rest}{\sum_{i=1}^{n} count(t,p_i)} \prec \frac{\sum_{i=1}^{n} T \cdot count(t,p_i)}{\sum_{i=1}^{n} count(t,p_i)} = T. \tag{3}$$

This conflicts with the assumption. Therefore, if a target is above or equal to threshold in a relation, it must always be above or equal to threshold at least in a partition p.

Theorem 1 is similar to pigeon hole's theory. According to theorem 1, we can prune targets below threshold in all partitions. That is, if $avg(t,p)$ meets T in a partition p, t will be included into candidates. Notice that this theorem is always satisfied without regard to how a relation is partitioned. In next section, we'll discuss how to partition a relation.

4 Partitioning Algorithms

In this section, we introduce BAP(*BAsic Partitioning*) algorithm, and propose POP(*POstponed Partitioning*) algorithm improved from BAP algorithm. These algorithms are different only in Gen_Partition procedure.

4.1 Common Parts of Algorithms BAP and POP

The basic idea of algorithms BAP and POP is to partition a relation logically to find candidates. These algorithms consist of two phases - the phase partitioning a relation and selecting candidates, and the phase computing exact average value of candidates. The common parts of partitioning algorithms are described as follows.

```
Algorithm Partitioning
input : Relation R, Threshold T
begin
   // Phase 1 : Partitioning and Selecting Candidates
   second_scan := false;
   bucket_num := 0;
   for all tuple d in R do
      if there is the bucket for d.target t in Counter C then
         Update(C,d);
      else
         if bucket_num < |C| then
            Insert(C,d);
            bucket_num++;
         else
            Gen_Partition(C,T,bucket_num);
            second_scan := true;
         end if
      end if
   end for

   // Phase 2 : Computing the Exact Value of Candidates
   if second_scan == false then
      Print_Results(C,T);
   else
      repeat
         read as many candidates as counter buckets to C;
         scan R and update C;
         Print_Results(C,T);
      until there are no more candidates;
   end if
end
```

In phase 1, a tuple d is read from relation R. The treatment of d varies according to whether or not the counter bucket for $d.target$ exists and whether or not there are any empty buckets in counter. If the bucket for $d.target$ exists, sum_val and cnt_val in the bucket will be updated. If the bucket for $d.target$ doesn't exist, but there are some empty buckets, the bucket for $d.target$ will be generated. If neighter the bucket for $d.target$ nor any empty buckets exist, Gen_Partition() will

be called. This procedure performs partitioning, which means selecting candidates, writing them, and resetting counter. However Gen_Partition() is different between in BAP and in POP. It'll be discussed more intensively in section 4.2 and 4.3. Such steps are repeated until R ends.

The phase 2 is to compute the exact average value of candidates. If the number of partitions is one, then all results can be returned from counter immediately. Otherwise, candidates are loaded into memory and R is scanned again to compute *sum_val* and *cnt_val* of candidates. If there are too many candidates for buckets, these steps will be over and over. Each scan will return some part of results.

4.2 Algorithm BAP

Procedure Gen_Partition. When an empty bucket is needed but doesn't exist, this procedure in BAP performs partitioning, which means selecting candidates, writing them, and resetting counter. According to theorem 1, *target*s, of which average values satisfy given threshold in a partition, are selected as candidates. These are written into disk, and counter is reset. The details are shown as follows.

```
Procedure Gen_Partition  // BAP algorithm
input : Counter C, Threshold T, bucket_num
begin
   write targets above T in C to disk;
   reset C;
   bucket_num := 0;
end
```

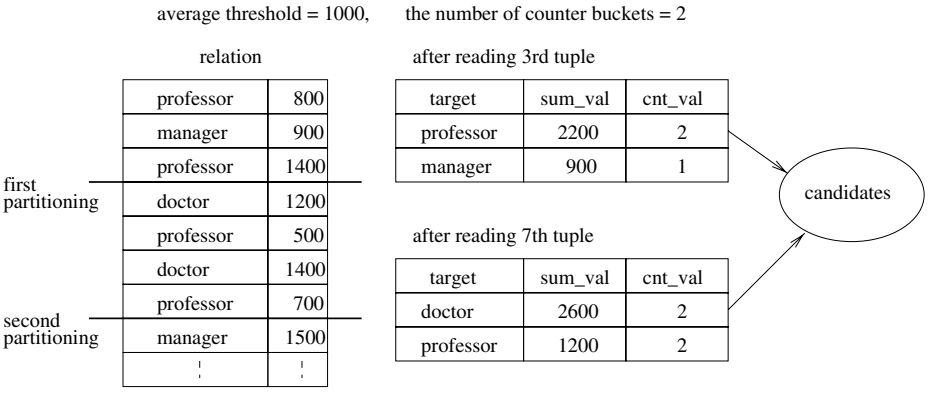


Fig. 3. BAP during Phase 1

Example. Fig. 3. shows an example of partitioning in BAP. When fourth tuple <doctor, 1200> is read, first partitioning takes place. Only the bucket for target {professor} satisfies threshold, so {professor} is written to disk. Then, all buckets are reset, new partition begins, and fourth tuple can be treated. These steps are repeated until the relation ends.

4.3 Algorithm POP

Procedure Gen_Partition. The idea of POP is to put off partitioning. When an empty bucket is needed but doesn't exist, all buckets below threshold are reset instead of writing candidates at once. Therefore, empty buckets increase and new tuple can be read again. If no bucket is reset, that is, *target*s in all buckets are candidates, then partitioning happens. The details are shown as follows.

```
Procedure Gen_Partition  // POP algorithm
input : Counter C, Threshold T, bucket_num
begin
   reset buckets below T in C;
   if there is no bucket to reset then
      write all targets in C to disk;
      reset C;
      bucket_num := 0;
   end if
end
```

Example. Fig. 4. shows an example of partitioning in POP. When fourth tuple is read, neither the bucket for {doctor} nor an empty bucket exists. So, {manager} bucket below threshold is reset. Now, it's possible to treat fourth tuple. Remember that new partition began this time in BAP. When eighth tuple is read, the bucket for {manager} doesn't exist in counter, there is no empty bucket, and all targets in counter meet threshold. Then, partitioning happens and all targets are written to disk.

4.4 Analysis

Now, we'll analyze BAP and POP. To postpone partitioning in POP affects the number of partition and the domain size of each partition. First, the number of partition in POP doesn't become larger than that in BAP. As the number of partition is smaller, the probability that peak data to appear decreases. Peak data mean that they are below threshold in a relation, but they meet threshold in a partition. The existences of peak data make candidates increase. The worst case of POP is when each partition is occupied with only peak data of which *cnt_val*s are one. Second, while the domain size of each partition except for the last is all the same in BAP, it isn't in POP. Because non-candidates are reset and new tuple can be treated as partitioning is put off.

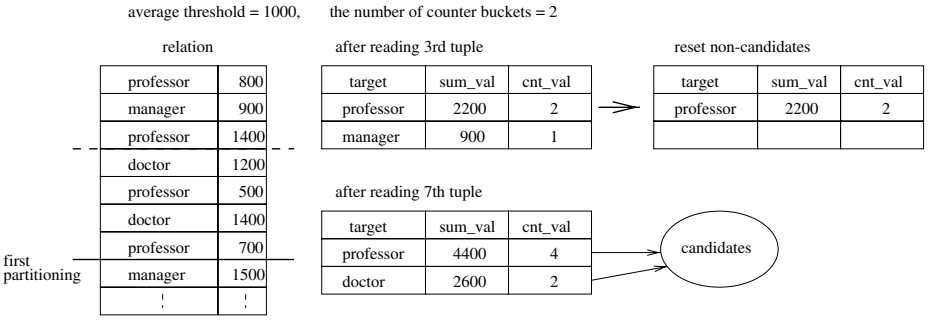


Fig. 4. POP during Phase 1

5 Experiments

Our experiments were run in an Ultrasparc-IIi 333MHz running Solaris 7, with 512MBs of RAM and 18GBs of local disk space. We implemented two algorithms in C++ and generated four datasets. To evaluate the relative efficiency of them, we varied memory size, data order or threshold about four datasets.

5.1 Datasets

All generated datasets consist of 100,000,000 tuples. These datasets are about 1.3GBs. The characteristics of each dataset rely on *target* distribution, *rest* distribution, or data order. The description of each dataset is as follows.

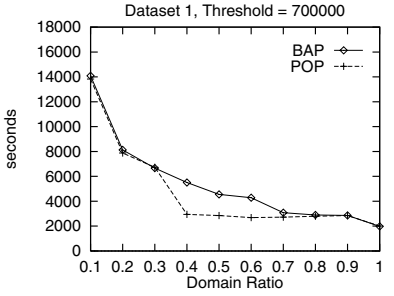
Dataset 1. The occurrences of *target*s follow normal distribution, and the values of *rest*s do uniform distribution. The domain size is 226,280. Max average value is about 999,000, and min is about 0.
Dataset 2. This was generated by sorting dataset 1 in order of *target*.
Dataset 3. The occurrences of *target*s follow uniform distribution, and the values of *rest*s do normal distribution. The domain size is 1,000,031. Max average value is about 21000, and min is about -19000.
Dataset 4. This was generated by sorting dataset 3 in order of *target*.

5.2 Memory Size

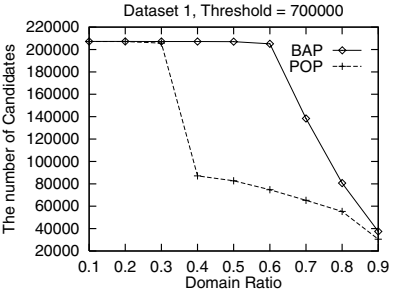
We defined as *domain ratio* a ratio of the number of counter buckets to domain size. *Domain ratio* ≥ 1.0 means counter buckets are a lot enough.

$$domain\ ratio = \frac{the\ number\ of\ counter\ buckets}{domain\ size}$$

The first experiments compared BAP with POP as we varied *domain ratio* from 0.1 to 1.0. The results are shown in Fig. 5. and Fig. 6. In Fig. 5(a) and

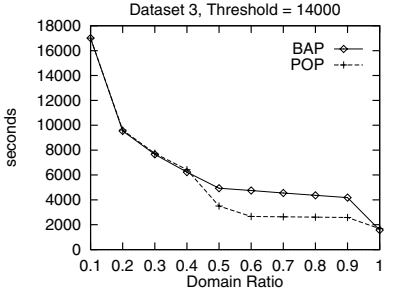


(a) Execution time

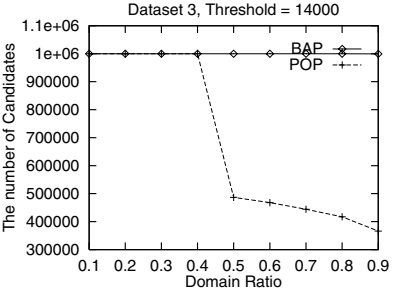


(b) Candidates

Fig. 5. Execution time and Candidates in Dataset 1

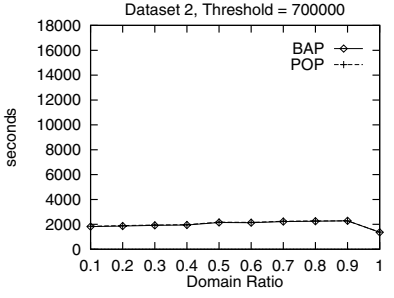


(a) Execution time

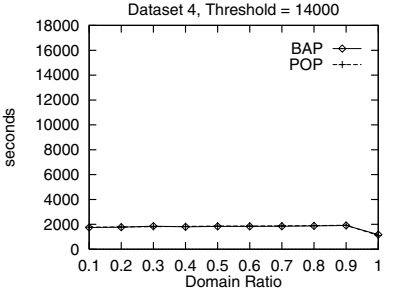


(b) Candidates

Fig. 6. Execution time and Candidates in Dataset 3

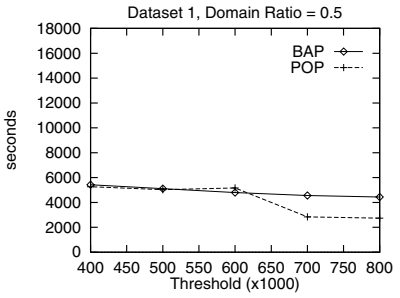


(a) Execution time

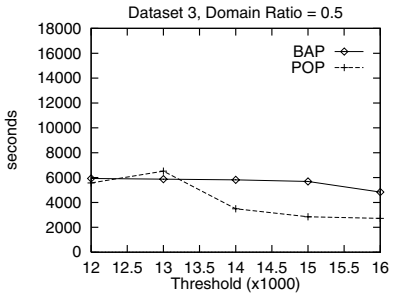


(b) Candidates

Fig. 7. Execution time vs. Data Order



(a) Dataset 1



(b) Dataset 3

Fig. 8. Execution time varing on Threshold

Fig. 6(a), it took about 2,000 seconds to scan the relation once. When *domain ratio* was from 0.4 or 0.5 to 0.9, AIQ were performed within two or three scans. POP was faster than BAP, and candidates were less in POP. When *domain ratio* was lower than 0.4, both BAP and POP played a role to find all unique data in phase 1.

The distribution of *target*s result in a few differences between Fig. 5(a) and Fig. 6(a). The tuples with the same *target* scattered more in dataset 3 than in dataset 1 because the occurrences of *target*s follow random distirbution in dataset 3. So, peak data happened more. The opposite cases about this matter are shown in section 5.3.

5.3 Data Order

The second experiments were for examining effects of data order. The results are shown in Fig. 7. These figures seem to be one line, but in fact two lines are overlapped. Without regard to memory size, AIQ were always performed with scanning the relation only two times. BAP and POP were almost the same. If these results are compared with Fig. 5(a). and Fig. 6(a)., they are excellent even if each dataset is the same except data order. This results from the exact selection of candidates because each target gathers together in dataset 2 and 4.

5.4 Threshold

The last experiments were for examining the relation between threshold and execution time. The results are shown in Fig. 8. When threshold was high in both Dataset 1 and Dataset 3, POP was performed with scanning the relation only two times. As threshold became lower, more scans were needed.

6 Conclusion

Until now, average functions have been out of consideration from iceberg query problem. In this paper, we introduced the theorem to select candidates, and presented BAP(*Basic Partitioning*) algorithm and POP(*Postponed Partitioning*) algorithm based on this theorem to compute average iceberg queries efficiently. These algorithms used the strategy to partition a relation logically. Moreover, POP algorithm used the technique to postpone partitioning.

Our experiments demonstrated that partitioning algorithms were affected by data order and memory size the most. If datasets are sorted, the performance is excellent without regard to memory size. Otherwise, memory size affects the performance very much. The comparison between two algorithms showed that POP was faster than BAP and candidates were smaller in POP when *domain ratio* was between 0.4 and 0.9, and they were almost the same when *domain ratio* was low.

References

1. M. Fang, N. Shivakumar, H. Garcia-Molina, R. Motwani, and J. D. Ullman, "Computing Iceberg Queries Efficiently", In Proc. of the 24th VLDB Conf., pages 299-310, 1998
2. R. T. Ng, L. V. S. Lakshmanan, J. Han, and A. Fang, "Exploratory Mining and Pruning Optimizations of Constrained Associations Rules", In Proc. of the ACM SIGMOD Conf. on Management of Data, pages 13-24, 1998
3. K. Beyer and R. Ramakrishnan, "Bottom-Up Computation of Sparse and Iceberg CUBEs", In Proc. of the ACM SIGMOD Conf., pages 359-370, 1999
4. A. Savasere, E. Omiecinski, and S. Navathe, "An Efficient Algorithm for Mining Association Rules in Large Databases", In Proc. of the 21st VLDB Conf., pages 432-444, 1995
5. S. Christodoulakis, "Multimedia Data Base Management: Applications and A Position Paper", In Proc. of the ACM SIGMOD Conf, pages 304-305, 1985
6. A. Ghafoor, "Multimedia Database Management System", Computing Surveys, Vol. 27, No. 4, pages 593-598, 1985
7. K. Whang, B. T. Vander-Zanden, and H. M. Taylor, "A Linear-time Probabilistic Counting Algorithms for DB Applications", ACM Transactions on Database Systems, 15(2):208-229, 1990
8. J. S. Park, M. S. Chen, and P. S. Yu, "An Effective Hash Based Algorithm For Mining Association Rules", In Proc. of ACM SIGMOD Conf., pages 175-186, 1995
9. R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules", In Proc. of the 20th VLDB Conf., pages 487-499, 1994
10. R. T. Ng and J. Han, "Efficient and Effective Clustering Method for Spatial Data Mining", In Proc. of the 20th VLDB Conf., pages 144-155, 1994
11. T. Zhang, R. Ramakrishnan, and M. Livny, "BIRCH: An Efficient Data Clustering Method for Very Large Databases", In Proc. of ACM SIGMOD Conf., pages 103-114, 1996
12. A. Bouju, A. Stockus, F. Bertrand, P. Boursier, "Client-Server Architecture for Accessing Multimedia and Geographic Databases within Embedded Systems", DEXA Workshop 1999

Security in Data Warehousing*

(Invited Talk)

Bharat Bhargava
CERIAS and Computer Science Department
Purdue University, West Lafayette,
Indiana 47906
bb@cs.purdue.edu

Data warehouse [2,4,5,6] is an integrated repository derived from multiple source (operational and legacy) databases. The data warehouse is created by either replicating the different source data or transforming them to new representation. This process involves reading, cleaning, aggregating and storing the data in the warehouse model. The software tools are used to access the warehouse for strategic analysis, decision-making, marketing types of applications. It can be used for inventory control of shelf stock in many departmental stores.

Medical and human genome researchers can create research data that can be either marketed or used by a wide range of users. The information and access privileges in data warehouse should mimic the constraints of source data. A recent trend is to create web-based data warehouses and multiple users can create components of the warehouse and keep an environment that is open to third party access and tools. Given the opportunity, users ask for lots of data in great detail. Since source data can be expensive,
its privacy and security must be assured. The idea of adaptive querying can be used to limit access after some data has been offered to the user. Based on the user profile, the access to warehouse data can be restricted or modified.

In this talk, I will focus on the following ideas that can contribute towards warehouse security.

1. Replication control

Replication can be viewed in a slightly different manner than perceived in traditional literature. For example, an old copy can be considered a replica of the current copy of the data. A slightly out-of date data can be considered as a good substitute for some users. The basic idea is that either the warehouse keeps different replicas of the same

* This research is partially supported by CERIAS and NSF under grant numbers 9805693-EIA, CCR-9901712.

Y. Kambayashi, M. Mohania, and A M. Tjoa (Eds.): DaWaK 2000, LNCS 1874, pp. 287-289, 2000.
© Springer-Verlag Berlin Heidelberg 2000

items or creates them dynamically. The legitimate users get the most consistent and complete copy of data while casual users get a weak replica. Such replica may be enough to satisfy the user's need but do not provide information that can be used maliciously or breach privacy. We have formally defined the equivalence of replicas [7] and this notion can be used to create replicas for different users. The replicas may be at one central site or can be distributed to proxies who may serve the users efficiently. In some cases the user may be given the weak replica and may be given an upgraded replica if wiling to pay or deserves it. Another idea related to this is the idea of witness that was discussed in [1].

2. Aggregation and Generalization

The concept of warehouse is based on the idea of using summaries and consolidators. This implies that source data is not available in raw form. This lends to ideas that can be used for security. Some users can get aggregates only over a large number of records where as others can be given for small data instances. The granularity of aggregation can be lowered for genuine users. The generalization idea can be used to give users a high level information at first but the lower level details can be given after the security constraints are satisfied. For example, the user may be given an approximate answer initially based on some generalization over the domains of the database. Inheritance is another notion that will allow increasing capability of access for users. The users can inherit access to related data after having access to some data item.

3. Exaggeration and Misleading

These concepts can be used to mutilate the data. A view may be available to support a particular query, but the values may be overstated in the view. For security concern, quality of views may depend on the user involved and user can be given an exaggerated view of the data. For example, instead of giving any specific sales figures, views may scale up and give only exaggerated data. In certain situations warehouse data can give some misleading information [3]; information which may be partially incorrect or difficult to verify the correctness of the information. For example, a view of a company's annual report may contain the net profit figure including the profit from sales of properties (not the actual sales of products).

4. Anonymity

Anonymity is to provide user and warehouse data privacy. A user does not know the source warehouse for his query and warehouse also does not know who is the user and what particular view a user is accessing (view may be constructed from many source databases for that warehouse). Note that a user must belong to the group of registered users and similarly, a user must also get data from only legitimate warehouses. In such cases, encryption and middleware are to be used to secure the connection between the users and warehouse so that no outside user (user who has not registered with the warehouse) can access the warehouse.

5. User Profile Based Security

User profile is a representation of the preferences of any individual user. User profiles can help in authentication and determining the levels of security to access warehouse data. User profile must describe how and what has to be represented pertaining to the users information and security level authorization needs. The growth in warehouses has made relevant information access difficult in reasonable time due to the large number of sources differ in terms of context and representation. Warehouse can use data category details in determining the access control. For example, if a new employee would like to access an unpublished annual company report, the warehouse server may deny access to it. The other alternative is to construct and return a view to her, which reflects only projected sales and profit. Such a construction of view may be transparent to the user. A server can also use the profile to decide whether the user should be given the access to associated graphical image with the data. The server has the option to reduce the resolution or the quality of images before making them available to users.

References

[1] Arnon Rosenthal and Edward Sciore, View Security as the Basis for Data Warehouse Security, Proceedings of the International Workshop on Design and Management of Data Warehouse (DMDW'2000), Sweden, June, 2000.

[2] Arun Sen and Varghese S. Jacob, Introduction – Guest Editorial, Communication of ACM, Vol. 41, No. 9, Sept. 1998.

[3] Demillo, R. A., D. P. Dobkin, R. J. Lipton, Even Databases that Lie can be Compromised, IEEE Transaction on Software Engineering, SE 5:1, pp. 73-75, 1978.

[4] Jim Scott, Warehousing Over the Web, Communication of ACM, Vol. 41, No. 9, Sept. 1998.

[5] Stephen R. Gardner, Building the Data Warehouse, Communication of ACM, Vol. 41, No. 9, Sept. 1998.

[6] Sunil Samtani, Mukesh Mohania, Vijay Kumar, Yahiko Kambayashi, Recent Advances and Research Problems in Data Warehousing, Advances in Database Technologies, ER Workshop, LNCS Vol. 1552. 1998.

[7] Meeliyal Annamalai and Bharat Bhargava, Defining Data Equivalence for Efficient Image Retrieval in a Distributed Environment, In Proceedings of the Second World Conference on Integrated Design and Process Technology, Austin, Texas, Dec. 1996.

Mining of Association Rules in Text Databases Using Inverted Hashing and Pruning

John D. Holt and Soon M. Chung *

Dept. of Computer Science and Engineering, Wright State University, Dayton, Ohio 45435, USA, {jholt,schung}@cs.wright.edu

Abstract. In this paper, we propose a new algorithm named Inverted Hashing and Pruning (IHP) for mining association rules between words in text databases. The characteristics of text databases are quite different from those of retail transaction databases, and existing mining algorithms cannot handle text databases efficiently, because of the large number of itemsets (i.e., words) that need to be counted. Two well-known mining algorithms, the Apriori algorithm [1] and Direct Hashing and Pruning (DHP) algorithm [5], are evaluated in the context of mining text databases, and are compared with the proposed IHP algorithm. It has been shown that the IHP algorithm has better performance for large text databases.

1 Introduction

Mining association rules in transaction databases has been demonstrated to be useful and technically feasible in several application areas [2, 3], particularly in retail sales. Let $\mathcal{I} = \{i_1, i_2, \ldots, i_m\}$ be a set of items. Let $\mathcal{D}$ be a set of transactions, where each transaction T is a set of items, such that $T \subseteq \mathcal{I}$. An association rule is an implication of the form $X \Rightarrow Y$, where $X \subset \mathcal{I}$, $Y \subset \mathcal{I}$, and $X \cap Y = \phi$. The association rule $X \Rightarrow Y$ holds in the database $\mathcal{D}$ with *confidence* c if $c\%$ of transactions in $\mathcal{D}$ that contain X also contain Y. The rule $X \Rightarrow Y$ has *support* s if $s\%$ of transactions in $\mathcal{D}$ contain $X \cup Y$. Mining association rules is to find all association rules that have support and confidence greater than user-specified minimum support (called *minsup*) and minimum confidence (called *minconf*) [1]. For example, beer and disposable diapers are items such that *beer* $\Rightarrow$ *diapers* is an association rule mined from the database if the co-occurrence rate of beer and disposable diapers (in the same transaction) is higher than *minsup* and the occurrence rate of diapers in the transactions containing beer is higher than *minconf*.

The first step in the discovery of association rules is to find each set of items (called *itemset*) that have co-occurrence rate above the minimum support. An itemset with at least the minimum support is called a *large itemset* or a *frequent itemset*. In this paper, the term *frequent itemset* will be used. The size of an itemset represents the the number of items contained in the itemset, and

* This research was supported in part by Ohio Board of Regents.

Y. Kambayashi, M. Mohania, and A M. Tjoa (Eds.): DaWaK 2000, LNCS 1874, pp. 290–300, 2000.
© Springer-Verlag Berlin Heidelberg 2000

an itemset containing k items will be called a k-itemset. For example, {beer, disposable diapers} can be a frequent 2-itemset. Finding all frequent itemsets is a very resource consuming task and has received a considerable amount of research effort in recent years. The second step of forming the association rules from the frequent itemsets is straightforward as described in [1]: For every frequent itemset f, find all non-empty subsets of f. For every such subset a, generate a rule of the form $a \Rightarrow (f - a)$ if the ratio of support$(f - a)$ to support(a) is at least *minconf*.

The association rules mined from point-of-sale (POS) transaction databases can be used to predict the purchase behavior of customers. In the case of text databases, there are several uses of mined association rules. The association rules for text can be used for building a statistical thesaurus. Consider the case that we have an association rule $B \Rightarrow C$, where B and C are words. A search for documents containing C can be expanded by including B. This expansion will allow for finding documents using C that do not contain C as a term. A closely related use is Latent Semantic Indexing, where documents are considered "close" to each other if they share a sufficient number of associations [4]. Latent Semantic Indexing can be used to retrieve documents that do not have any terms in common with the original text search expression by adding documents to the query result set that are "close" to the documents in the original query result set.

The word frequency distribution of a text database can be very different from the item frequency distribution of a sales transaction database. Additionally, the number of unique words in a text database is significantly larger than the number of unique items in a transaction database. Finally, the number of unique words in a typical document is much larger than the number of unique items in a transaction. These differences make the existing algorithms, such as the Apriori [1] and Direct Hashing and Pruning (DHP) [5], ineffective in mining association rules in the text databases.

A new algorithm suitable for the mining association rules in text databases is proposed in this paper. The algorithm is named Inverted Hashing and Pruning (IHP), and is described in Section 3. The results of the performance analysis are discussed in Section 4. The new algorithm demonstrated significantly better performance than Apriori and DHP algorithms for large text databases.

2 Text Databases

Traditional domains for finding frequent itemsets, and subsequently the association rules, include retail point-of-sale (POS) transaction database and catalog order database [2]. The number of items in a typical POS transaction is well under a hundred. The mean number of items and distribution varies considerably depending upon the retail operation. In the referenced papers that provided experimental results, the number of items per transaction ranged from 5 to 20.

The word distribution characteristics of text data present some scale challenges to the algorithms that are typically used in mining transaction databases. A sample of text documents was drawn from the 1996 TReC [9] data collection.

The sample consisted of the April 1990 Wall Street Journal articles that were in the TReC collection. There were 3,568 articles and 47,189 unique words. Most of those words occur in only a few of the documents. Some of the key distribution statistics are:

48.2% of the words occur in only one document;
13.4% of the words occur in only two documents;
7.0% of the words occur in only 3 documents.

The mean number of unique words in a document was 207, with a standard deviation of 174.2 words. In this sample, only 6.8% of the words occurred in more than 1% of the documents. A single day sample of January 21, 1990 was taken as well. In that sample, there were 9,830 unique words, and 78.3% of the words occurred in three or fewer documents.

The characteristics of this word distribution have profound implications for the efficiency of association rule mining algorithms. The most important implications are: (1) the large number of items and combinations of items that need to be counted; and (2) the large number of items in each document in the database.

It is commonly recognized in the information retrieval community that words that appear uniformly in a text database have little value in differentiating documents, and further those words occur in a substantial number of documents [6]. It is reasonable to expect that frequent itemsets composed of highly frequent words (typically considered to be words with occurrence rates above 20%) would also have little value. Therefore, text database miners will need to work with itemsets composed of words that are not too frequent, but are frequent enough. The range of minimum and maximum support suitable for word association mining is not known at this time.

The experiments described in this paper were conducted as part of a study for building a thesaurus from association rules. The thesaurus study was interested in discovering associations between words in fairly close proximity in the text. The documents were broken into fragments of about 200 words each. The word frequency distribution of the document fragments was similar to the frequency distribution reported above, though not as skewed. Some key distribution statistics are:

29.3% of the words occur in only one fragment;
21.6% of the words occur in only two fragments;
9.7% of the words occur in only 3 fragments.

The mean number of unique words in a document fragment was 101, with a standard deviation of 46.4 words. In this sample, only 3.5% of the words occurred in more than 1% of the document fragments.

3 Inverted Hashing and Pruning (IHP) Algorithm

Inverted Hashing and Pruning (IHP) is analogous to the Direct Hashing and Pruning (DHP) [5] in the sense that both use hash tables to prune some of the candidate itemsets. In DHP, during the k-th pass on the database, every $(k+1)$-itemset within each transaction is hashed into a hash table, and if the count of the hash value of a $(k+1)$-itemset is less than the minimum support, it is not

considered as a candidate in the next pass. In IHP, for each item, the transaction identifiers (TIDs) of the transactions that contain the item are hashed to a hash table associated with the item, which is named TID Hash Table (THT) of the item. When an item occurs in a transaction, the TID of this transaction is hashed to an entry of the THT of the item, and the entry stores the number of transactions whose TIDs are hashed to that entry. Thus, the THT of each item can be generated as we count the occurrences of each item during the first pass on the database. After the first pass, we can remove the THTs of the items which are not contained in the set of frequent 1-itemsets, F_1, and the THTs of the frequent 1-itemsets can be used to prune some of the candidate 2-itemsets. In general, after each pass $k \geq 1$, we can remove the THT of each item that is not a member of any frequent k-itemset, and the remaining THTs can prune some of the candidate $(k+1)$-itemsets.

Consider a transaction database with seven items; A, B, C, D, E, F, and G. Figure 1 shows the THTs of these items at the end of the first pass. In our example, each THT has five entries for illustration purpose. Here we can see that item D occurred in five transactions. There were two TIDs hashed to 0, one TID hashed to 1, and two TIDs hashed to 4.

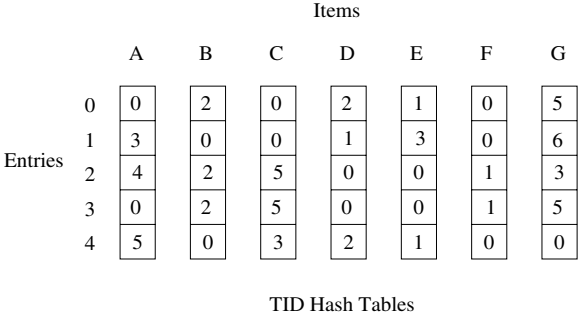


Fig. 1. TID Hash Tables at the end of the first pass

If the minimum support count is 7, we can remove the THTs of the items B, D, E, and F as shown in Figure 2. Only the items A, C, and G are frequent and are used to determine C_2, the set of candidate 2-itemsets. Based on the Apriori algorithm, {A,C}, {A,G}, and {C,G} are generated as candidate 2-itemsets by paring the frequent 1-itemsets. However, in IHP, we can eliminate {A,G} from consideration by using the THTs of A and G. Item A occurs in 12 transactions, and item G occurs in 19 transactions. However, according to their THTs, they can co-occur in at most 6 transactions. Item G occurs in 5 transactions whose TIDs identifiers are hashed to 0, and item A occurs in no transactions that have such TIDs. Thus, none of those 5 transactions that contain G also contains A. Item A occurs in 3 transactions whose TIDs are hashed to 1 and item G occurs in 6 transactions with those TIDs. So, in the set of transactions whose TIDs are

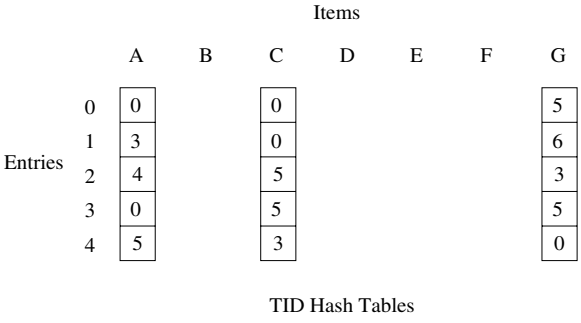


Fig. 2. TID Hash Tables of frequent 1-itemsets

hashed to 1, items A and G can co-occur at most 3 times. The other THT entries corresponding to the TID hash values of 2, 3, and 4 can be examined similarly, and we can determine items A and G can co-occur in at most 6 transactions, which is below the minimum support level.

In general, for a candidate k-itemset, we can estimate its maximum possible support count by adding the minimum count of the k items at each entry of their THTs. If the maximum possible support count is less than the required minimum support, it is pruned from the candidate set.

The details of the IHP algorithm are presented below:

1) $Database$ = set of transactions;
2) $Items$ = set of items;
3) transaction = $\langle TID,\ \{x \mid x \in Items\}\rangle$;
4) **Comment:** F_1 is a set of frequent 1-itemsets
5) $F_1 = \phi$;
6) **Comment:** Read the transactions and count the occurrences of each item and create a TID Hash Table (THT) for each item using a hash function h
7) **foreach** transaction $t \in Database$ **do begin**
8) **foreach** item x in t **do begin**
9) $x.count++$;
10) $x.THT[(h(t.TID)]++$;
11) **end**
12) **end**
13) **Comment:** Form the set of frequent 1-itemsets
14) **foreach** item $i \in Items$ **do**
15) **if** $i.count/|Database| \geq minsup$
16) **then** $F_1 = F_1 \cup i$;
17) **Comment:** Find F_k, the set of frequent k-itemsets, where $k \geq 2$
18) **for** $(k = 2; F_{k-1} \neq \phi; k++)$ **do begin**
19) **foreach** item $i \notin$ any frequent $(k-1)$-itemset **do**
20) remove $i.THT$;

21) **Comment:** C_k is the set of candidate k-itemsets
22) $C_k = \phi$;
23) $tempC_k = \phi$;
24) **Comment:** $F_{k-1} * F_{k-1}$ is a natural join of
        F_{k-1} and F_{k-1} on the first $k-2$ items
25) **foreach** $x \in \{F_{k-1} * F_{k-1}\}$ **do**
26)   **if** $\neg\exists y \mid y = (k-1)$-subset of $x \wedge y \notin F_{k-1}$
27)   **then** $tempC_k = tempC_k \cup x$;
28) **Comment:** Prune the candidate k-itemsets using the THTs
29) **foreach** $x \in tempC_k$ **do**
30)   if $GetMaxPossibleCount(x)/|Database| \geq minsup$
31)   **then** $C_k = C_k \cup x$;
32) **Comment:** Scan the transactions to count candidate k-itemsets
33) **foreach** transaction $t \in Database$ **do begin**
34)   **foreach** k-itemset x in t **do**
35)   **if** $x \in C_k$ **then** $x.count++$;
36) **end**
37) **Comment:** F_k is the set of frequent k-itemsets
38) $F_k = \phi$;
39) **foreach** $x \in C_k$ **do**
40)   **if** $x.count/|Database| \geq minsup$
41)   **then** $F_k = F_k \cup x$;
42) **end**
43) $Answer = \cup_k F_k$;

The formation of the set of candidate itemsets can be done effectively when the items in each itemset are stored in a lexical order, and itemsets are also lexically ordered. As specified in line 25, candidate k-itemsets, for $k \geq 2$, are obtained by performing the natural join operation $F_{k-1} * F_{k-1}$ on the first $k-2$ items of the $(k-1)$-itemsets in F_{k-1} assuming that the items are lexically ordered in each itemset [1]. For example, if F_2 includes {A, B} and {A, C}, then {A, B, C} is a potential candidate 3-itemset. Then the potential candidate k-itemsets are pruned in line 26 by using the property that all the $(k-1)$-subsets of a frequent k-itemset should be frequent $(k-1)$-itemsets [1]. Thus, for {A, B, C} to be a candidate 3-itemset, {B, C} also should be a frequent 2-itemset. To count the occurrences of the candidate itemsets efficiently as the transactions are scanned, they can be stored in a hash tree, where the hash value of each item occupies a level in the tree [1].

$GetMaxPossibleCount(x)$ returns the maximum number of transactions that may contain k-itemset x by using the THTs of the k items in x. Let's denote the k items in x by $x[1], x[2], \ldots, x[k]$. Then $GetMaxPossibleCount(x)$ can be defined as follows:

$GetMaxPossibleCount$(itemset x)
begin
    $k = size(x)$;
    MaxPossibleCount = 0;
    for $(j = 0; j < size(THT); j++)$ **do**
        MaxPossibleCount += $min(x[1].THT[j], x[2].THT[j], \ldots, x[k].THT[j])$;
    return (MaxPossibleCount)
end

$size(x)$ represents the number of items in an itemset x and $size(THT)$ represents the number of entries in the THT.

The Partition algorithm [7] used the list of TIDs of the transactions containing each item, called a TID-list. Once the TID-list is generated for each item by scanning the database once, we don't need to scan the database again, because the support count of any candidate itemset can be obtained by intersecting the TID-lists of the individual items in the candidate itemset. However, a major problem is that the size of the TID-lists would be too large to maintain when the number of transactions in the database is very large. On the other hand, the number of entries in the THTs can be limited to a certain value, and it has been shown that even a small number of entries can effectively prune many candidate itemsets.

For further performance improvement, the IHP algorithm can be used together with the transaction trimming and pruning method proposed as a part of the DHP algorithm [5]. The concept of the transaction trimming and pruning is follows: During the k-th pass on the database, if an item is not a member of at least k candidate k-itemsets within a transaction, it can be removed from the transaction for the next pass (transaction trimming), because it cannot be a member of a candidate $(k + 1)$-itemset. Moreover, if a transaction doesn't have at least $k + 1$ candidate k-itemsets, it can be removed from the database (transaction pruning), becuase it cannot have a candidate $(k + 1)$-itemset.

4 Performance Analysis of the IHP algorithm

Some performance tests have been done with the IHP algorithm. The first objective was to assess the effect of the TID Hash Table (THT) on the performance. The second objective was to assess the impact of transaction trimming and pruning upon performance. To meet these objectives, we studied the performance of four miners, Apriori, DHP, IHP without transaction trimming and pruning (IHPwoTTP), and IHP with transaction trimming and pruning (IHPwTTP). All four of these miners were derived from the same code base. All of the test runs were made on a 400 MHz Pentium-II machine with 384 Mbytes of memory. All of the miners were written in Java, and the IBM 1.1.7A JVM was used. The JVM memory for objects was constrained (via the *mx* parameter) to 256 Mbytes. The initial heap size was set to 256 Mbytes (via the *ms* parameter) to control the

effect of heap growth overhead on performance. For the IHP, the TID hash table (THT) size was 400 entries, and the hash table size for DHP was 500,000 entries for the runs comparing the four miners. Two databases of different sizes were used for the experiments. The minimum support level used in these tests was 1% of the text fragments, and words with occurrence rates higher than 20% were treated as stop words and ignored.

The small data collection consists of two weeks of the Wall Street Journal with 1,738 documents broken into 5,954 fragments with a file size of 5.7 Mbytes. There were 1,643 frequent 1-itemsets, 11,641 frequent 2-itemsets, and 2,621 frequent 3-itemsets.

In Figure 3, we find that for the small collection of text documents, DHP and both IHP algorithms outperform the Apriori algorithm. In this test, there were only 3 passes on the database. The Apriori algorithm evaluated 1,607,124 candidate itemsets, the DHP algorithm evaluated 1,359,819 candidate itemsets, and the IHP algorithms evaluated 654,049 candidate itemsets. Compared to the Apriori, the DHP was able to achieve a 15% reduction in the number of candidate itemsets considered, and the IHP was able to achieve nearly a 60% reduction.

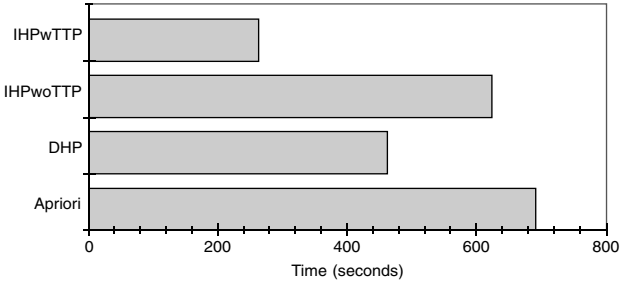


Fig. 3. 3 passes on 5,954 document fragments and 1,643 frequent items

In Figure 4, we can see a similar performance pattern for a larger data collection. This data collection consisted of three months of the Wall Street Journal in 34,828 document fragments with a file size of 34.5 Mbytes. There were 1,618 frequent items, 11,140 frequent 2-itemsets, and 1,925 frequent 3-itemsets.

The efficiency of pruning the candidate itemsets is depicted in Figure 5. The Apriori algorithm counted 1,308,153 candidate 2-itemsets in the second pass. The DHP algorithm, with a hash table of 500,000 entries, counted 1,291,461 candidate 2-itemsets, which is about 1.3% reduction over Apriori. The IHP algorithms counted 820,071 candidate 2-itemsets, which is about 37.3% reduction over Apriori.

Figure 6 compares the processing times of the four mining algorithms on a pass by pass basis. The DHP algorithm requires slightly more processing time in the first and second passes due to the overhead of generating the hash table during each pass. In the second pass, the two IHP algorithms require substantially less processing time than the DHP and Apriori algorithms. This is due to

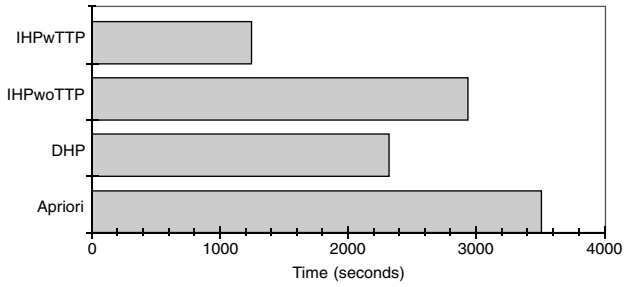


Fig. 4. 3 passes on 34,828 document fragments and 1,618 frequent items

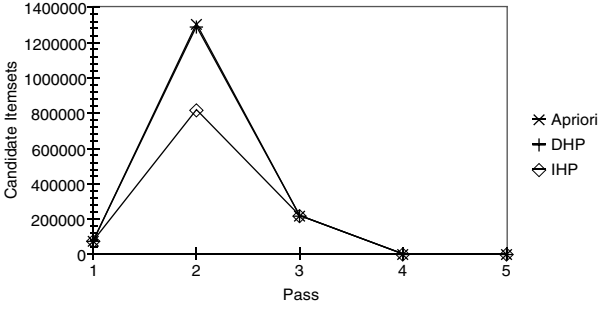


Fig. 5. Candidates counted in each pass

the reduction in the number of candidate 2-itemsets. In the third pass, all four algorithms counted the same number of candidate 3-itemsets, and as a result the IHP without the transaction trimming and pruning (IHPwoTTP) takes the same amount of time as the Apriori. However, the IHP with transaction trimming and pruning (IHPwTTP) is faster than the DHP. On the third pass, both algorithms have the same number of candidate itemsets. The difference in performance is due to the transaction trimming and pruning performed in the second pass.

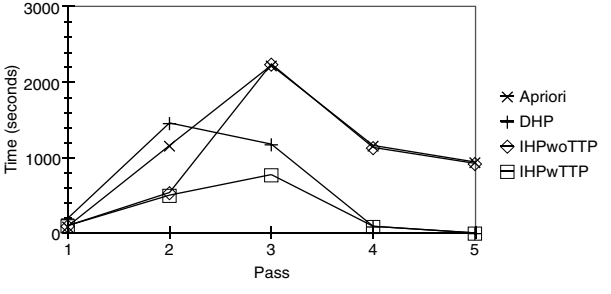


Fig. 6. Comparison of pass times

During the second pass, both DHP and IHPwTTP used the transaction trimming and pruning and rewrote the transactions with the candidate 2-itemsets. As a result, each transaction had an average of 1,449 candidate 2-itemsets for DHP, and an average of 1,324 candidate 2-itemsets for IHPwTTP. Thus, in the third pass, the DHP algorithm had more 3-itemsets to check against the set of candidate 3-itemsets than IHPwTTP. On the fourth and subsequent passes, both IHPwTTP and DHP processed with the same number of candidates and the same database size, so there is no big difference in their processing times.

We evaluated the effect of the TID hash table (THT) size on the performance of IHP. Figure 7 shows the total processing time of the IHPwoTTP for three passes on the larger test database. The zero entry in the THT corresponds to the case of Apriori, and it is included for a comparison purpose. The total processing time of IHPwoTTP decreases as the size of the THT increases up to 400 entries, because a larger THT can prune more candidate itemsets. However, as the THT size increases, we need more memory space to store the THTs, and there are more THT entries to check in order to determine whether a candidate itemset can be pruned or not. In our test, when each THT has 500 entries, the total number of candidate itemsets was reduced further, but the speedup from this reduction was not sufficient to overcome the increase in processing time required to handle the additional THT entries. We also tested the sensitivity of the DHP performance to its hash table size. Doubling the hash table size to 1,000,000 entries resulted only about 9% reduction in total processing time for the same test case.

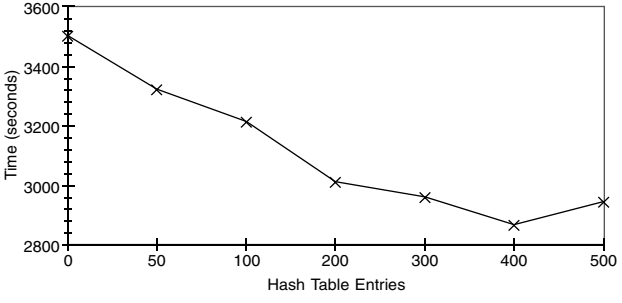


Fig. 7. Effect of TID hash table size on the IHP processing time

5 Conclusions

This study has two main results. First, the study shows that the IHP algorithm is more effective for mining text databases than the existing Apriori and DHP algorithms. Secondly, the study shows that transaction trimming and pruning is an effective technique to use when mining text databases and has a synergistic effect when combined with reducing the number of candidate itemsets.

The IHP algorithms can effectively use available memory in the first pass to sharply reduce the number of candidate itemsets in the second pass. Since a lot of TID hash tables are pruned between the first and second passes, most of the memory used for holding the TID hash tables in the first pass is available to hold the candidate itemsets for the second and subsequent passes. In large part, this effect is a result of the distribution of word occurrences discussed in Section 2. The large number of words with very low occurrence rates results in a situation that the preponderance of the TID hash tables generated in the first pass are pruned prior to the initiation of the second pass.

Transaction trimming and pruning reduces the work required to scan the transactions. This reduction is significant for text database processing, and is a significant part of the total processing time reduction provided by the DHP algorithm and the IHP with transaction trimming and pruning algorithm. The beneficial effect of transaction trimming and pruning for text data mining was found to have a larger effect than the candidate pruning alone.

The candidate pruning provided by the IHP algorithm combines synergistically with the improvement from transaction trimming and pruning to provide a superior level of performance.

References

1. R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules," *Proc. of the 20th VLDB Conf.*, 1994, pp. 487–499.
2. S. Brin, R. Motwani, J. Ullman, and S. Tsur, "Dynamic Itemset Counting and Implication Rules for Market Basket Data," *Proc of the ACM SIGMOD Int'l Conf. on Management of Data*, 1997, pp. 255–264.
3. M. S. Chen, J. Han, and P. S. Yu, "Data Mining: An Overview from a Database Perspective," *IEEE Trans. on Knowledge and Data Engineering*, Vol. 8, No. 6, Dec. 1996, pp. 866–883.
4. M. Gordon and S. Dumais, "Using Latent Semantic Indexing for Literature Based Discovery," *Journal of the Amer. Soc. of Info Science*, Vol. 49, No. 8, June 1998, pp. 674–685.
5. J. S. Park, M. S. Chen, and P. S. Yu, "Using a Hash-Based Method with Transaction Trimming for Mining Association Rules," *IEEE Trans. on Knowledge and Data Engineering*, Vol. 9, No. 5, Sep/Oct 1997, pp. 813–825.
6. G. Salton, *Automatic Text Processing: the transformation, analysis, and retrieval of information by computer*, Addison-Wesley Publishing, 1988.
7. A. Savasere, E. Omiecinski, and S. Navathe, "An Efficient Algorithm for Mining Association Rules in Large Databases," *Proc. of the 21st VLDB Conf.*, 1995, pp. 432–444.
8. H. Toivonen, "Sampling Large Databases for Association Rules," *Proc. of the 22nd VLDB Conf.*, 1996, pp. 134–145.
9. E. M. Voorhees and D. K. Harmon (editors), *The Fifth Text Retrieval Conference*, National Institute of Standards and Technology, 1997.

SQL Based Association Rule Mining Using Commercial RDBMS (IBM DB2 UDB EEE)

Takeshi Yoshizawa, Iko Pramudiono, Masaru Kitsuregawa

Institute of Industrial Science, The University of Tokyo 7-22-1 Roppongi, Minato-ku, Tokyo 106, Japan
IBM Japan Co.,Ltd. 1-1, Nakase, Mihama-ku, Chiba-shi, Chiba 261-8522, Japan

{yoshi,iko,kitsure}@tkl.iis.u-tokyo.ac.jp

Abstract. Data mining is becoming increasingly important since the size of databases grows even larger and the need to explore hidden rules from the databases becomes widely recognized. Currently database systems are dominated by relational database and the ability to perform data mining using standard SQL queries will definitely ease implementation of data mining. However the performance of SQL based data mining is known to fall behind specialized implementation and expensive mining tools being on sale. In this paper we present an evaluation of SQL based data mining on commercial RDBMS (IBM DB2 UDB EEE). We examine some techniques to reduce I/O cost by using View and Subquery. Those queries can be more than 6 times faster than SETM SQL query reported previously. In addition, we have made performance evaluation on parallel database environment and compared the performance result with commercial data mining tool (IBM Intelligent Miner). We prove that SQL based data mining can achieve sufficient performance by the utilization of SQL query customization and database tuning. Keywords: data mining, parallel SQL, query optimization, commercial RDBMS

1 Introduction

Extracting valuable rules from a large set of data has attracted lots of attention from both researcher and business community. This is particularly driven by explosion of the information amount stored in databases such as Data Warehouses during recent years. In business world, many organizations begin to apply data mining techniques directly to raw transaction data and some results such as unidentified buying patterns and credit card fraud indications are widely recognized.

One method of data mining is finding association rule [1]. Basket data analysis is typical of this method. There are some approaches proposed to mine association rules, [1,2,6,9] some of them are based on relational database standard SQL [3,7,8]. But this kind of mining is known as CPU power demanding application and it has to handle very large amounts of transaction data. Unfortunately SQL approach is reported to

Y. Kambayashi, M. Mohania, and A M. Tjoa (Eds.): DaWaK 2000, LNCS 1874, pp. 301–306, 2000.
© Springer-Verlag Berlin Heidelberg 2000

have drawback in performance although it has many advantages such as seamless integration with existing system and high portability.[7]

On the other hand recently most major commercial database systems have included capabilities to support parallelization although no report available about how the parallelization affects the performance of complex query required by association rule mining. This fact motivated us to examine how efficiently SQL based association rule mining can be parallelized and speeded up using commercial parallel database system (IBM DB2 UDB EEE). We propose two techniques to enhance association rule mining query based on SETM [3]. And we have also compared the performance with commercial mining tool (IBM Intelligent Miner). Our performance evaluation shows that we can achieve comparable performance with commercial mining tool using only 4 nodes.

Some considerable works on effective SQL queries to mine association rule such as [7] and [8] didn't examine the effect of parallelization. [4] and [5] have reported a performance evaluation on PC cluster as parallel platform. Comparison with natively coded programs is also reported. However we use currently available commercial products for the evaluation.

2 Association Rule Mining Based on SQL

An example of association rule mining is finding "if a customer buys A and B then 90% of them buy also C" in transaction databases of large retail organizations. This 90% value is called confidence of the rule. Another important parameter is support of an itemset, such as support ({A,B,C}), which is defined as the percentage of the itemset contained in the entire transactions. For above example, confidence can also be measured as support ({A,B,C}) divided by support ({A,B}).

In our experiments we employed three type of SQL query. First of all the standard SQL query using SETM algorithm [3]. Second is the enhanced SQL query using view materialization technique. Third is another enhanced SQL query using subquery technique.

2.1 SQL query using SETM algorithm

Transaction data is transformed into the first normal form (transaction ID, item). In the first pass we simply gather the count of each item. Items that satisfy the minimum support are inserted into large itemsets table C_1 that takes form (item, item count). SETM employs temporary tables to reuse item combination in next pass. In first pass, transactions that match large itemsets are preserved in temporary table R_1.

In other passes for example pass k, we first generate all lexicographically ordered candidate itemsets of length k into another temporary table RTMP_k by self-joining table R_(k-1) that contains k-1 length transaction data. Then we generate the count for those itemsets, itemsets that meet minimum support are included into large itemset table C_k. Finally transaction data R_k of length k generated by matching items in candidate itemset table RTMP_k with items in large itemsets. In order to avoid excessive I/O, we disable the log during the execution.

2.2 Enhanced SETM query using view materialize technique

SETM has to materialize its temporary tables namely R_k and RTMP_k. Those temporary tables are only required in the next pass and they are not needed for generating the rules. In fact, those tables can be deleted after execution of its subsequent pass. Based on this observation we could avoid materialization cost of those temporary tables by replacing the table creation with view.

2.3 Enhanced SETM query using subquery technique

We expect significant performance improvement with utilization of view, however view still requires time to access the system catalog and are holding locks to the system catalog table during creating views so we further use subquery instead of temporary tables. Therefore we embed the generation of item combinations into the query to generate large itemsets.

3 Performance Evaluation

3.1 Parallel Execution Environment

In our experiment we employed commercial Parallel RDBMS: IBM DB2 UDB EEE version 6.1 on IBM UNIX Parallel Server System: IBM RS/6000 SP. 12 nodes make this system and using shared nothing achitecture. Each node has POWER2 77Mhz CPU, 4.4GB SCSI hard disk, 256MB RAM and connected by High Performance Switch HPS with 100MB/s network speed.

We also used commercial data mining tool IBM Intelligent Miner on single node of RS/6000 SP for performance comparison with the SQL based data mining.

3.2 Dataset

We use synthetic transaction data generated with program described in Apriori algorithm paper [2] for experiment. The parameters used are : number of transactions 200000, average transaction length 10 and number of items 2000. Transaction data is partitioned uniformly by hashing algorithm corresponds to transaction ID among processing nodes' local hard disks.

3.3 Performance comparison

Figure.1 shows the execution time on each degree of parallelization. On average, we can derive that View and Subquery SQL is about 6 times faster than SETM SQL regardless of the number of nodes. The result is also compared with the execution time of Intelligent Miner on single processing node. It is true that Intelligent Miner on single node with transaction data stored in flat file is much faster than the SQL queries. However, the View and Subquery SQL are 50% faster than Intelligent Miner

on single node if the transaction data have to be read from RDBMS. We exemplified that we can achieve comparable performance of Intelligent Miner on single node with flat file by activating only 4 nodes when we used View and Subquery SQL. The result gives evidence for the effectiveness of parallelization of SQL query to mine association rule.

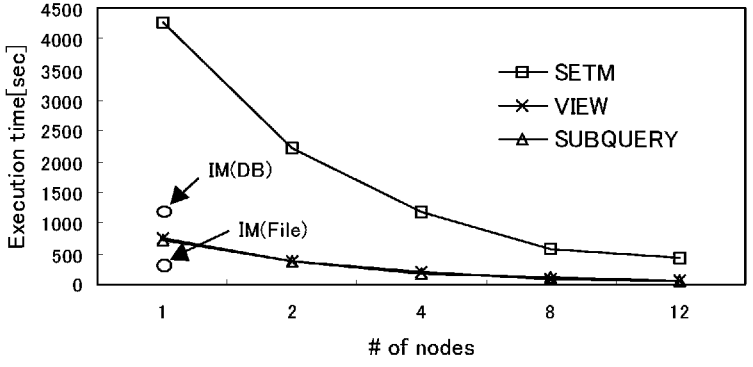


Fig. 1. Execution time on parallel database environment

The speedup ratio is shown in figure 2. This is also reasonably good, especially View and Subquery SQL are not being saturated as the number of processing nodes increased. That means they can be parallelized well. The execution is 11 times faster with 12 nodes. In parallel environment, network potentially becomes botleneck which degrades the speed-up ratio. However our experiments suggest that association rule mining using variants of SETM is mostly CPU bound and network I/O is negligible.

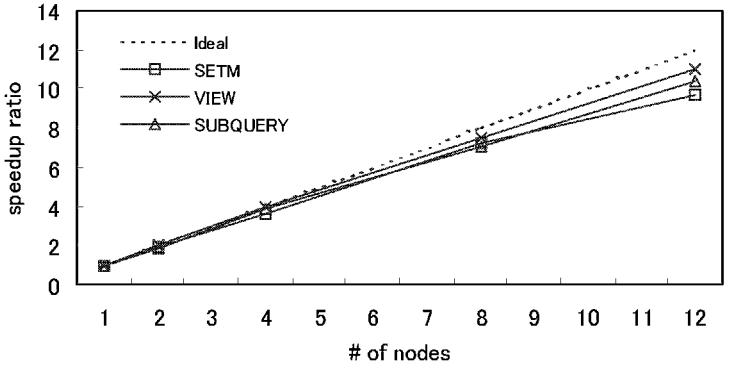


Fig. 2. Speedup ratio in parallel database environment

3.4 Analysis of execution trace

In this section, we compared the three variations of SQL query described before. The performance evaluation is done on 12 nodes.

The mining is two passes long. It is well known that in most cases the second pass generates huge amount of candidate itemsets thus it is the most time consuming phase[4][5]. Our results are very much alike. Almost over 80% of execution time belongs to PASS2 in all three SQL queries. Obviously View and Subquery SQL complete their first and second passes faster than SETM SQL query.

We have recorded the execution traces of the three SQL in each PASS. The decomposition of execution time is analysed as shown Figure 3 (PASS2) respectively. Comparing the elapsed time with the cpu time at Fig 3, we find that both are close for View SQL and Subquery SQL. This means these SQL's are cpu bound, while SETM SQL is not cpu bound. Most of execution time of SETM query is dominated by disk write time for creating temporary table such as R_k and RTMP_k. We can also see that sort time is almost equal for all three SQL's, which represents the cost of group by aggregation.

In PASS2, SETM reuses item combinations in temporary table R1 on the secondary storage that is generated in PASS1. We replace it with view or subquery. Then data is transferred directly through memory from PASS1 to PASS2. Figure 3 indicates that PASS2 of those modified SQL queries only read data from buffer pool. Thus the disk write time of View SQL and Subquery SQL is almost negligible, although it is dominait for SETM SQL. This analysis clarifies the problem of SETM and how to cost can be reduced for View and Subquery SQLs, which is the key to the performance improvement.

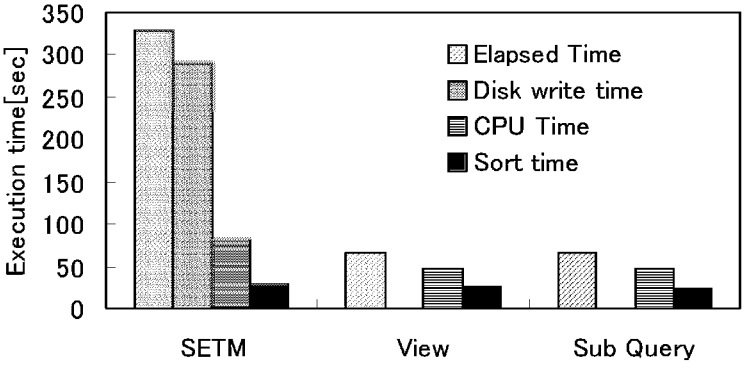


Fig. 3. Decomposition of execution time of PASS2 for three types of SQL queries

4 Summary and Conclusion

The ability to perform data mining using standard SQL queries will benefit data warehouses with the better integration with commercial RDBMS. It also allows easier porting codes among different systems.

In this paper, we reported the parallelization of SQL query to mine association rule on commercial RDBMS (IBM DB2 UDB EEE). We showed that good speedup ratio can be achieved, that means it is parallelized well. We also examined two variations of SETM SQL queries to improve performance, which reduce I/O cost by using View materialize or Subquery technique, and can achieve performance more than 6 times faster than SETM SQL query with two passes. We expect still more improvement with more passes. We'd like to report it next time.

We have compared the parallel implementation of SQL based association rule mining with commercial data mining tool (IBM Intelligent Miner). Through real implementation, we have showed our improved SETM query using View or Subquery can beat the performance of specialized tool with only 4 nodes while original SETM query needs more than 24 processing nodes to achieve the same performance. We don't have to buy expensive data mining tool, since parallel execution of SQL comes at no extra cost. It is also extremely easy to implement and flexible.

There remain lots of further investigations. We'd like to check the performance with more passes situations. In addition we plan to do using more large transaction data. And we'd like to investigate the effect of intra parallelism under SMP environment.

References

1. R. Agrawal, T. Imielinski, A. Swami. Mining Association Rules between Sets of Items in Large Databases. In Proc. of the ACM SIGMOD Conference on Management of Data, 1993.
2. R. Agrawal, R. Strikant. Fast Algorithms for Mining Association Rules. In Proc. of the Very Large Database (VLDB) Conference, 1994.
3. M. Houtsma, A. Swami. Set-oriented Mining of Association Rules. In Proc. of International Conference on Data Engineering (ICDE), 1995.
4. Iko Pramudiono, Takahiko Shintani, Takayuki Tamura, Masaru Kitsuregawa. Parallel SQL Based Association Rule Mining on Large Scale PC Cluster: Performance Comparison with Directly Coded C Implementation. In Proc. of Third Pacific-Asia Conference on Knowledge Discovery a nd Data Mining (PAKDD99), 1999.
5. Iko Pramudiono, Takahiko Shintani, Takayuki Tamura, Masaru Kitsuregawa. Mining Generalized Association Rule using Parallel RDB Engine on PC Cluster. In Proc. of First International Conference on Data Warehousing and Knowledege Discovery (DAWAK99), 1999.
6. Jong Soo Park, Ming-Syan Chen, Philip S. Yu. An Effective Hash-Based Algorithm for Mining Association Rules. In Proc. of the ACM SIGMOD Conference on Management of Data, 1995.
7. Sunita Sarawagi, Shiby Thomas, Rakesh Agrawal. Integrating Association Rule Mining with Relational Database Systems : Alternatives and Implications. In Proc. of the ACM SIGMOD Conference on Management of Data, 1998.
8. Shiby Thomas, Sharma Chakravarthy. Performance Evaluation and Optimization of Join Queries for Association Rule Mining. In Proc. of First International Conference on Data Warehousing and Knowledege Discovery (DAWAK99), 1999.
9. T. Shintani, M. Kitsuregawa. Parallel Mining Algorithms for Generalized Association Rules with Classification Hierarchy. In Proc. of the ACM SIGMOD Conference on Management of Data, 1998.

On Supporting Interactive Association Rule Mining

Bart Goethals and Jan Van den Bussche

Limburg University, Belgium
http://www.luc.ac.be/theocomp/

Abstract. We investigate ways to support interactive mining sessions, in the setting of association rule mining. In such sessions, users specify conditions (filters) on the associations to be generated. Our approach is a combination of the incorporation of filtering conditions inside the mining phase, and the filtering of already generated associations. We present several concrete algorithms and compare their performance.

1 Introduction

The interactive nature of the mining process has been acknowledged from the start [3]. It motivated the idea of a "data mining query language" [5–8, 10] and was stressed again by Ng, Lakshmanan, Han and Pang [11]. A data mining query language allows the user to ask for specific subsets of association rules. Efficiently supporting data mining query language environments is a challenging task.

In this paper, working in the concrete setting of association rule mining, we consider a class of conditions on associations to be generated which should be expressible in any reasonable data mining query language: Boolean combinations of atomic conditions, where an atomic condition can either specify that a certain item occurs in the body of the rule or the head of the rule, or set a threshold on the support or on the confidence. A *mining session* then consists of a sequence of such Boolean combinations (henceforth referred to as *queries*).

We present the first algorithm to support interactive mining sessions efficiently. We measure efficiency in terms of the total number of itemsets that are generated, but do not satisfy the query, and the number of scans over the database that have to be performed. Specifically, our results are the following:

1. The filtering achieved by exploiting the query conditions is *non-redundant*, in the sense that it never generates an itemset that, apart from the minimal support and confidence thresholds, could give rise to a rule that does not satisfy the query. Therefore, the number of generated itemsets during the execution of a query, becomes proportional to the strength of the conditions in the query: the more specific the query, the faster its execution.
2. Not only is the number of passes through the database reduced, but also the size of the database itself, again proportionally to the strength of the conditions in the query.

Y. Kambayashi, M. Mohania, and A M. Tjoa (Eds.): DaWaK 2000, LNCS 1874, pp. 307–316, 2000.
© Springer-Verlag Berlin Heidelberg 2000

3. A generated itemset will, within a session, never be regenerated as a candidate itemset: results of earlier queries are reused when answering a new query.

The idea that filters can be integrated in the mining algorithm was initially launched by Srikant, Vu, and Agrawal [12], who considered filters that are Boolean expressions over the presence or absence of certain items in the rules (filters specifically as bodies or heads were not discussed). The algorithms proposed in their paper are not optimal: they generate and test several itemsets that do not satisfy the filter, and their optimizations also do not always become more efficient for more specific filters.

Also Lakshmanan, Ng, Han and Pang worked on the integration of constraints on itemsets in mining, considering conjunctions of conditions such as those considered here, as well as others (arbitrary Boolean combinations were not discussed) [9, 11]. Of the various strategies for the so-called "CAP" algorithm they present, the one that can handle the filters considered in the present paper is their "strategy II." Again, this strategy generates and tests itemsets that do not satisfy the filter. Also their algorithms implement a rule-filter by separately mining for possible heads and for possible bodies, while we tightly couple filtering of rules with filtering of sets.

Both works do not discuss the reuse of results acquired from earlier queries within a session.

This paper is further organized as follows. We assume familiarity with the notions and terminology of association rule mining and the Apriori algorithm [1, 2]. In Section 2, we present a way of incorporating query-constraints inside a frequent set mining algorithm. In Section 3, we discuss ways of supporting interactive mining sessions. In Section 4, we present an experimental evaluation of our algorithms, and discuss their implementation.

2 Exploiting Constraints

As already mentioned in the Introduction, the constraints we consider in this paper are Boolean combinations of atomic conditions. An atomic condition can either specify that a certain item i occurs in the body of the rule or the head of the rule, denoted respectively by Body(i) or Head(i), or set a threshold on the support or on the confidence.

In this section, we explain how we can incorporate these constraints in the mining algorithm. We first consider the special case of constraints where only conjunctions of atomic conditions or their negations are allowed.

2.1 Conjunctive Constraints

Let $b_1, \ldots, b_\ell$ be the items that must be in the body by the constraint; $b'_1, \ldots, b'_{\ell'}$ those that must not; $h_1, \ldots, h_m$ those that must be in the head; and $h'_1, \ldots, h'_{m'}$ those that must not.

Recall that an association rule $X \Rightarrow Y$ is only generated if $X \cup Y$ is a frequent set. Hence, we only have to generate those frequent sets that contain every b_i and h_i, plus some of the subsets of these frequent sets that can serve as bodies or heads. Therefore we will create a set-filter corresponding to the rule-filter, which is also a conjunctive expression, but now over the presence or absence of an item i in a frequent set, denoted by $\text{Set}(i)$ and $\neg\text{Set}(i)$. We do this as follows:

1. For each positive literal $\text{Body}(i)$ or $\text{Head}(i)$ in the rule-filter, add the literal $\text{Set}(i)$ in the set-filter.
2. If for an item i both $\neg\text{Body}(i)$ and $\neg\text{Head}(i)$ are in the rule-filter, add the negated literal $\neg\text{Set}(i)$ to the set-filter.
3. Add the minimal support threshold to the set-filter.
4. All other literals in the rule-filter are ignored because they do not restrict the frequent sets that must be generated.
   Formally, the following is readily verified:

Lemma 1. *An itemset Z satisfies the set-filter if and only if there exists itemsets X and Y such that $X \cup Y = Z$ and the rule $X \Rightarrow Y$ satisfies the rule-filter, apart from the confidence threshold.*

So, once we have generated all sets Z satisfying the set-filter, we can generate all rules satisfying the rule-filter by splitting all these Z in all possible ways in a body X and a head Y such that the rule-filter is satisfied. Lemma 1 guarantees that this method is "sound and complete".

We thus need to explain two things:

1. Finding all frequent Z satisfying the set-filter.
2. Finding, for each Z, the frequencies of all bodies and heads X and Y such that $X \cup Y = Z$ and $X \Rightarrow Y$ satisfies the rule-filter.

Finding the frequent sets satisfying the set-filter. Let $Pos := \{i \mid \text{Set}(i)$ in set-filter$\}$ and $Neg := \{i \mid \neg\text{Set}(i)$ in set-filter$\}$. Note that $Pos = \{b_1, \ldots, b_\ell, h_1, \ldots, h_m\}$. Denote the dataset of transactions by $\mathcal{D}$. We define the following derived dataset $\mathcal{D}_0$:

$$\mathcal{D}_0 := \{t - (Pos \cup Neg) \mid t \in \mathcal{D} \text{ and } Pos \subseteq t\}$$

In other words, we ignore all transactions that are not supersets of Pos and from all transactions that are not ignored, we remove all items in Pos plus all items that are in Neg.

We observe: (proof omitted)

Lemma 2. *Let p be the absolute support threshold defined in the filter. Let $\mathcal{S}_0$ be the set of itemsets over the new dataset $\mathcal{D}_0$, without any further conditions, except that their support is at least p. Let $\mathcal{S}$ be the set of itemsets over the original dataset $\mathcal{D}$ that satisfy the set-filter, and whose support is also at least p. Then*

$$\mathcal{S} = \{s \cup Pos \mid s \in \mathcal{S}_0\}.$$

We can thus perform any frequent set generation algorithm, using only $\mathcal{D}_0$ instead of $\mathcal{D}$. Note that the size of $\mathcal{D}_0$ is exactly the support of Pos in $\mathcal{D}$. Still put differently: we are mining in a world where itemsets that do not satisfy the filter simply do not exist. The correctness and optimality of our method is thus automatically guaranteed.

Note however that now an itemset I, actually represents the itemset $I \cup Pos$! We thus head-start with a lead of k, where k is the cardinality of Pos, in comparison with standard, non-filtered mining.

Finding the frequencies of bodies and heads. We now have all frequent sets containing every b_i and h_i, from which rules that satisfy the rule-filter can be generated. Recall that in phase 2 of Apriori, rules are generated by taking every item in a frequent set as a head and the others as body. All heads that result in a confident rule, with respect to the minimal confidence threshold, can then be combined to generate more general rules. But, because we now only want rules that satisfy the filter, a head must always be a superset of $\{h_1, \ldots, h_m\}$ and must not include any of the h'_i and b_i (the latter because bodies and heads of rules are disjoint). In this way, we again head-start with a lead of m. Similarly, a body must always be a superset of $\{b_1, \ldots, b_\ell\}$ and may not include any of the b'_i and h_i.

The following lemma tells us that these potential heads and bodies are already present, albeit implicitly, in $\mathcal{S}_0$:

Lemma 3. *Let $\mathcal{S}_0$ be as in Lemma 2. Let $\mathcal{B}$ ($\mathcal{H}$) be the set of bodies (heads) of those association rules over $\mathcal{D}$ that satisfy the rule-filter. Then*

$$\mathcal{B} = \{s \cup \{b_1, \ldots, b_\ell\} \mid s \in \mathcal{S}_0 \text{ and } s \cap \{b'_1, \ldots, b'_{\ell'}, h_1, \ldots, h_m\} = \emptyset\}$$

and

$$\mathcal{H} = \{s \cup \{h_1, \ldots, h_m\} \mid s \in \mathcal{S}_0 \text{ and } s \cap \{h'_1, \ldots, h'_{m'}\, b_1, \ldots, b_\ell\} = \emptyset\}.$$

So, for the potential bodies (heads), we use, in $\mathcal{S}_0$, all sets that do not include any of the b'_i and h_i (h'_i and b_i), and add all b_i (h_i). Hence, all we have to do is to determine the frequencies of these subsets in one additional pass. (We do not yet have these frequencies because these sets do not contain either items b_i or h_i, while we ignored transactions that do not contain all items b_i and h_i.)

Each generated itemset can thus have up to three different "personalities:"

1. A frequent set that satisfies the set-filter;
2. A frequent set that can act as body of a rule that satisfies the rule-filter;
3. Analogously for a head.

We finally generate the desired association rules from the appropriate sets, on condition that they have enough confidence.

Example. We illustrate our method with an example. Assume we are given the rule-filter

$$\text{Body}(1) \wedge \neg\text{Body}(2) \wedge \text{Head}(3) \wedge \neg\text{Head}(4)$$
$$\wedge \neg\text{Body}(5) \wedge \neg\text{Head}(5) \wedge \text{support} \geq 1 \wedge \text{confidence} \geq 50\%.$$

We begin by converting it to the set-filter

$$\text{Set}(1) \wedge \text{Set}(3) \wedge \neg\text{Set}(5) \wedge \text{support} \geq 1.$$

Hence $Pos = \{1, 3\}$ and $Neg = \{5\}$. Consider a database consisting of the three transactions $\{2, 3, 5, 6, 9\}$, $\{1, 2, 3, 5, 6\}$ and $\{1, 3, 4, 8\}$. We ignore the first transaction because it is not a superset of Pos. We remove items 1 and 3 from the second transaction because they are in Pos, and we also remove 5 because it is in Neg. We only remove items 1 and 3 from the third transaction. After reading, according to Lemmas 1 and 2, the two resulting transactions, one of the itemsets we find in $\mathcal{S}_0$ is $\{4, 8\}$, which actually represents the set $\{1, 3, 4, 8\}$. It also represents a potential body, namely $\{1, 4, 8\}$, but it does not represent a head, because it includes item 4, which must not be in the head according to the given rule-filter. As another example, the empty set now represents the set $\{1, 3\}$ from which a rule can be generated. It also represents a potential body and a potential head.

2.2 Arbitrary Boolean Filters

Assume now given a rule-filter that is an arbitrary Boolean combination of atomic conditions. We can put it in Disjunctive Normal Form[1] and then generate all frequent itemsets for every disjunct (which is a conjunction) in parallel by feeding every transaction of the database to every disjunct, and processing them there as described in the previous subsection.

However, this approach is a bit simplistic, as it might generate some sets and rules multiple times. For example, consider the following filter: $\text{Body}(1) \vee \text{Body}(2)$. If we convert it to its corresponding set-filter (disjunct by disjunct), we get $\text{Set}(1) \vee \text{Set}(2)$. Then, we would generate for both disjuncts all supersets of $\{1, 2\}$. We can avoid this problem by putting the set-filter in *disjoint* DNF.[2] Then, no itemset can satisfy more than one set-disjunct. On the other hand, this does not solve the problem of generating some *rules* multiple times. Consider the equivalent disjoint DNF of the above set-filter: $\text{Set}(1) \vee (\text{Set}(2) \wedge \neg\text{Set}(1))$. The first disjunct thus contains the set $\{1, 2\}$ and all of its supersets. If we generate for every itemset all potential bodies and heads according to every rule-disjunct, both rule-disjuncts will still generate all rules with the itemset $\{1, 2\}$ in the body. The easiest way to avoid this problem is to put already the rule-filter in disjoint DNF.

[1] Any Boolean expression has an equivalent DNF.

[2] In disjoint DNF, the conjunction of any two disjuncts is unsatisfiable. Any boolean expression has an equivalent disjoint DNF.

Until now, we have disregarded the possible presence of negated thresholds in the filters, which can come from the conversion to disjoint DNF, or from the user himself. Due to space limitations, we defer their treatment to the full paper.

3 Interactive Mining

3.1 Integrated Filtering or Post-Processing?

In the previous section, we have seen a way to integrate filter conditions tightly into the mining of association rules. We call this *integrated filtering*. At the other end of the spectrum we have *post-processing*, where we perform standard, non-filtered mining, save the resulting itemsets and rules, and then query those results for the filter.

Integrated filtering has obvious advantages over post-processing:

However, as already mentioned in the Introduction, data mining query language environments must support an interactive, iterative mining process, where a user repeatedly issues new queries based on what he found in the answers of his previous queries. Now consider a situation where minimal support requirements and data set particulars are favorable enough so that post-processing is not infeasible to begin with. Then the global, non-filtered mining operation, on the result of which the filtering will be performed by post-processing, *can be executed once and its result materialized for the remainder of the data mining session* (or part of it).

In that case, if the session consists of, say, 20 data mining queries, these 20 queries amount to standard retrieval queries on the materialized mining results. In contrast, answering every single of the 20 queries by an integrated filter will involve at least 20, and often many more, passes over the data, as each query involves a separate mining operation. We can analyze the situation easily as follows.

Consider a session in which the user issues a total of m data mining queries over a database of size n. Suppose that the total number of association rules (given a minimal support and confidence requirement) over these data equals r. Let t be the time required to generate all these rules. Moreover, it is not unreasonable to estimate that in post-processing, each filter executes in time proportional to r, and that in integrated filtering, each filter executes in time proportional to n. Then the total time spent by the post-processing approach is $t + m \cdot r$, while in the integrated filtering approach this is $m \cdot n$. Hence, if $n > r$, we have proved the following:

Proposition 1. *The integrated filtering total time is guaranteed to grow beyond the post-processing total time after exactly* $m = \lceil t/(n-r) \rceil$ *queries.*

3.2 Online Filtering: Basic Approach

From the above discussion it is clear that we should try to combine the advantages of integrated filtering and post-processing. We now introduce such an approach, which we call *online filtering*.

In the online approach, all rules and itemsets that result from every posed query, as well as all intermediate generated itemsets, are saved incrementally. Initially, when the user issues his first query, nothing has been mined yet, and thus we answer it using integrated filtering.

Every subsequent query is first converted to its corresponding rule- and set-filter in disjoint DNF. For every disjunct in the set-filter, the system checks all currently saved itemsets. If an itemset satisfies the disjunct, it is added to the data structure holding itemsets, that is used for mining that disjunct, as well as all of its subsets that satisfy the disjunct (note that these subsets may not all be saved; if they are not, we have to count their supports during the first scan through the dataset). We also immediately add all candidate sets, and if they were already saved, we add their support, so that they need not to be regenerated and recounted.

If no new candidate sets can be generated, this means that all necessary sets were already saved, and we are done. However, if this is not the case, we can now begin our filtered mining algorithm with the important generalization that in each iteration, candidate itemsets of different cardinalities are now generated. In order for this to work, candidate itemsets that turn out to be non-frequent must be kept so that they are not regenerated in later iterations. This generalization was first used by Toivonen in his sampling algorithm [13].

Saving all generated itemsets and rules gives us another advantage that can be exploited by the integrated filtering algorithm. Consider a set-filter stating that item 1 and 2 must be in the set. In the first pass of the algorithm all single itemsets are generated as candidate sets over the new dataset $\mathcal{D}_0$ (cf. Section 2.1). We explained that these single itemsets actually represent supersets of $\{1, 2\}$. Normally, before we generate a candidate set, we check if all of its subsets are frequent. Of course, this is impossible if these subsets do not even exist in $\mathcal{D}_0$. Now, however, we can check in the saved results for a subset with too low support; if we find this, we can avoid generating the candidate.

For rule generation, the same techniques apply. We thus obtain an algorithm which reuses previously generated itemsets and rules as if they had been generated in previous iterations of the algorithm. We are optimal in the sense that we never generate and test sets or rules that were generated before.

We also note that techniques for dealing with main memory overflow in the setting of standard, non-filtered, non-interactive association rule mining [14], remain valid in our approach.

3.3 Online Filtering: Improvements

In the worst case, the saved results do not contain anything that can be reused for answering a filter, and hence the time needed to generate the rules that satisfy the filter is equal to the time needed when answering that filter using the integrated filtering approach. In the best case, all requested rules are already saved, and hence the time needed to find all rules that satisfy the filter is equal to the time needed for answering that filter using post-processing. In the average case, part of the needed sets and rules are saved and will then be used to speed

up the integrated filtering approach. If the time gained by this speedup is more than the time needed to find the reusable sets and rules, then the online approach will always be faster than the integrated filtering approach. In the limit, all rules will be materialized, and hence all subsequent filters will be answered using post-processing.

Could it be that the time gained by the speedup in the integrated filtering is less than the time needed to find the reusable sets and rules? This could happen when a lot of sets and rules are already saved, but almost none of them satisfies the filter. We can however counter this phenomenon by improving the speedup. The improvement is based on estimating what is currently saved, as follows.

We keep track of a set-filter ϕ_{sets} which describes the saved sets, and of a rule-filter ψ_{rules} which describes the saved rules. Both filters are initially *false*. Given a new query (rule-filter) ψ the system now goes through the following steps: (step 3 was described in section 2.1)

1. $\psi_{mine} := \psi \wedge \neg\psi_{rules}$
2. $\psi_{rules} := \psi_{rules} \vee \psi$
3. Convert the rule-filter ψ_{mine} to the set-filter ϕ
4. $\phi_{mine} := \phi \wedge \neg\phi_{sets}$
5. $\phi_{sets} := \phi_{sets} \vee \phi$

After this, we perform:

1. Generate all frequent sets according to ϕ_{mine}, using the basic online approach.
2. Retrieve all saved sets satisfying $\phi \wedge \neg\phi_{mine}$.
3. Add all needed subsets that can serve as bodies or heads.
4. Use ψ_{mine} to generate rules from the sets of steps 1 and 2.
5. Retrieve all saved rules satisfying ψ.

Note that the filter ϕ_{mine} is much more specific than the original filter ϕ. We thus obtain the improvement in speedup from integrated filtering, which we already pointed out to be proportional to the strength of the filter.

3.4 Avoiding Exploding Filters

The improvement just described incurs a new problem. The formula ψ_{rules} (or ϕ_{sets}) becomes longer with the session. When, given the next filter ψ, we mine for $\psi \wedge \neg\psi_{rules}$, we will convert to disjoint DNF which could explode.

To avoid this, consider ψ_{rules} in DNF: $\psi_1 \vee \cdots \vee \psi_n$. Instead of the full filter $\psi \wedge \neg\psi_{rules}$, we are going to use a filter $\psi \wedge \neg\psi'_{rules}$, where ψ'_{rules} is obtained from ψ_{rules} by keeping only the least restrictive disjuncts ψ_i (their negation will thus be most restrictive). In this way $\psi \wedge \neg\psi'_{rules}$ is kept short.

But how do we measure restrictiveness of a ψ_i? Several heuristics come to mind. A simple one is to keep for each ψ_i the number of saved sets that satisfy it. These numbers can be maintained incrementally.

4 Experimental Comparison

For our experiments, we have implemented an extensively optimized version of the Apriori algorithm, equipped with the filtering optimizations as described in the previous sections.

We experimented with a session of 40 queries using the integrated filtering approach, the post-processing approach and the online approach. The transaction database was synthetically generated using the program provided by the Quest research group at IBM Almaden and contained 1 000 000 transaction over 10 000 items. The performance figures are shown in Figure 1.

The first 20 queries all require different items in the rules such that the online approach is just a little faster than the integrated filtering approach, because it cannot reuse that much pre-generated sets and rules. From there on the online approach has already collected a lot of sets and rules that can be reused. After the 20th query we can see some improvement on this until the 30th query, where the online approach has collected all sets and rules needed to answer further queries and hence the time needed to answer these queries is equal to the time needed to answer these queries using the post-processing approach.

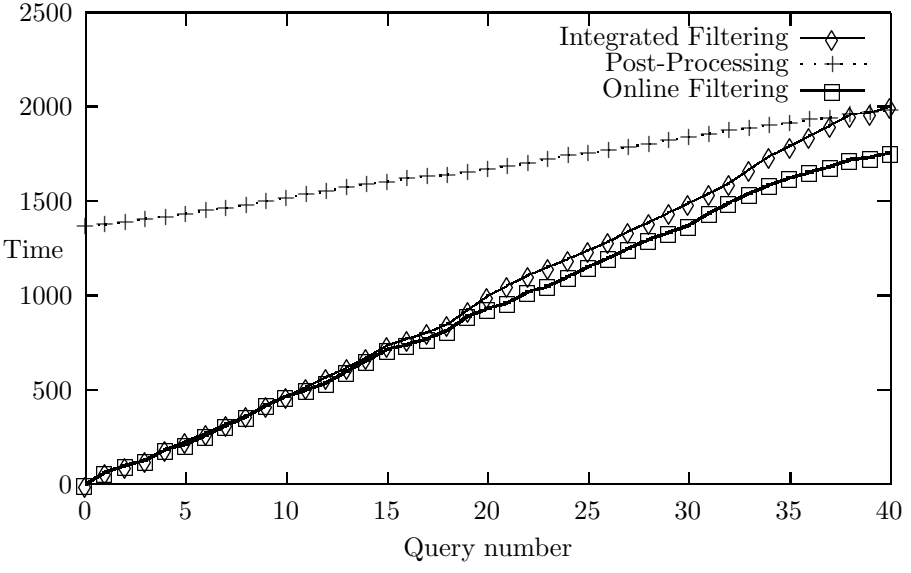


Fig. 1. Experiments. Time is in seconds.

References

1. R. Agrawal, T. Imielinski, and A.N. Swami. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, volume 22:2 of *SIGMOD Record*, pages 207–216. ACM Press, 1993.
2. R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A.I. Verkamo. Fast discovery of association rules. In Fayyad et al. [4], pages 307–328.
3. U.M. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. From data mining to knowledge discovery: An overview. In Fayyad et al. [4], pages 1–34.
4. U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors. *Advances in Knowledge Discovery and Data Mining*. MIT Press, 1996.
5. J. Han, Y. Fu, K. Koperski, W. Wang, and O. Zaiane. DMQL: A data mining query language for relational databases. Presented at SIGMOD'96 Workshop on Research Issues on Data Mining and Knowledge Discovery.
6. J. Han, Y. Fu, W. Wang, et al. DBMiner: A system for mining knowledge in large relational databases. In E. Simoudis, J. Han, and U. Fayyad, editors, *Proceedings 2nd International Conference on Knowledge Discovery & Data Mining*, pages 250–255. AAAI Press, 1996.
7. T. Imielinski and H. Mannila. A database perspective on knowledge discovery. *Communications of the ACM*, 39(11):58–64, 1996.
8. T. Imielinski and A. Virmani. MSQL: A query language for database mining. *Data Mining and Knowledge Discovery*, 3(4):373–408, December 1999.
9. L.V.S. Lakshmanan, R.T. Ng, J. Han, and A. Pang. Optimization of constrained frequent set queries with 2-variable constraints. In A. Delis, C. Faloutsos, and S. Ghandeharizadeh, editors, *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, volume 28:2 of *SIGMOD Record*, pages 157–168. ACM Press, 1999.
10. R. Meo, G. Psaila, and S. Ceri. An extension to SQL for mining association rules. *Data Mining and Knowledge Discovery*, 2(2):195–224, June 1998.
11. R.T. Ng, L.V.S. Lakshmanan, J. Han, and A. Pang. Exploratory mining and pruning optimizations of constrained association rules. In L.M. Haas and A. Tiwary, editors, *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, volume 27:2 of *SIGMOD Record*, pages 13–24. ACM Press, 1998.
12. R. Srikant, Q. Vu, and R. Agrawal. Mining association rules with item constraints. In D. Heckerman, H. Mannila, and D. Pregibon, editors, *Proceedings 3rd International Conference on Knowledge Discovery & Data Mining*, pages 66–73. AAAI Press, 1997.
13. H. Toivonen. Sampling large databases for association rules. In T. M. Vijayaraman, Alejandro P. Buchmann, C. Mohan, and Nandlal L. Sarda, editors, *Proceedings 22th International Conference on Very Large Data Bases*, pages 134–145. Kaufmann, 1996.
14. Y. Xiao and M.H. Dunham. Considering main memory in mining association rules. In M. K. Mohania and A. Min Tjoa, editors, *Data Warehousing and Knowledge Discovery*, volume 1676 of *Lecture Notes in Computer Science*, pages 209–218. Springer, 1999.

Discovering Temporal Patterns for Interval-based Events

Po-shan Kam and Ada Wai-chee Fu

Department of Computer Science and Engineering
The Chinese University of Hong Kong
{pskam,adafu}@cse.cuhk.edu.hk

1 Introduction

In many daily transactions, the time when an event takes place is known and stored in databases. Examples range from sales records, stock exchange, patient records, to scienti c databases in geophysics and astronomy. Such databases incorporate the concept of time which describes when an event starts and ends as historical records [9]. The temporal nature of data provides us with a better understanding of the trend or pattern over time. In market-basket data, we can have a pattern like "75% of customers buy peanuts when butter *starts* to be in big sales and *before* bread is sold out". We observe that there may be some correlations among peanuts, butter and bread so that we can have better planning for marketing strategy. Knowledge discovery in temporal databases thus catches the attention of researchers [8, 4].

Previous work for knowledge discovery in temporal data includes [2, 6, 5]. These techniques only treat data as series in chronological order and mainly support point-based events. Therefore, the physical ordering of events would be quite simple and the expressive power in specifying temporal relations such as *during, overlaps*, etc. is limited. When we deal with temporal data for events which last over a period of time, besides parallel and serial ordering of event sequences, we may nd more interesting temporal patterns. For instance, patterns like "event A occurs *during* the time event B happens" or "event A's occurrence time *overlaps* with that of event B and both of these events happen *before* event C appears" cannot be expressed as simple sequential orders. On the other hand, temporal logic [7] is suggested to be used for expressing temporal patterns de- ned over categorical data. Temporal operators such as *since, until* and *next* are used. We may have patterns like "event A always occurs *until* event B appears". Simple ordering of events is considered.

In this paper, we consider interval-based events where the duration of events is expressed in terms of endpoint values, and these are used to form temporal constraint in the discovery process. We introduce the notion of temporal representation which is capable of expressing the relationships between interval-based events. We develop new methods for nding such interesting patterns.

Y. Kambayashi, M. Mohania, and A M. Tjoa (Eds.): DaWaK 2000, LNCS 1874, pp. 317–326, 2000.
© Springer-Verlag Berlin Heidelberg 2000

2 Problem Statement

Suppose we are given a temporal database $\mathcal{D}$, which stores a list of clinical records. Each record contains a person-id, name of disease and a pair of ordered time points t_s and t_e where $t_s \leq t_e$ and both are integers. They are the **start time** and **end time**, indicating the period during which the patient contracted certain kind of disease.

A record may have other attributes, for simplicity, we consider here a single temporal attribute and denote it as an **event**. We assume a set **E** of event types. An event E has an associated time of occurrence and it is speci ed by a triple (A, t_s, t_e), where $A \in \mathbf{E}$ is an event type and t_s and t_e are the start time and the end time, respectively. We also use $E.t_s$ and $E.t_e$ to indicate these times.

We assume that each event is associated with one person. A **sequence** of events is de ned as a list of events where each event is associated with the same person such that for person j, we have the following sequence s_j, where $s_j = \langle (A_1, t_{s_1}, t_{e_1}), (A_2, t_{s_2}, t_{e_2}) ... (A_n, t_{s_n}, t_{e_n}) \rangle$. The events are ordered by the end times where $t_{e_i} \leq t_{e_{i+1}}$ for all $i = 1, ..., n-1$.

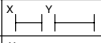
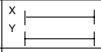
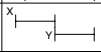
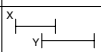
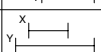
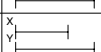
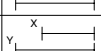
Relation	Relation for Inverse	Pictorial Example	Endpoints Constraint
X before Y	Y after X		$X.t_e < Y.t_s$
X equal Y	Y equal X		$X.t_s = Y.t_s$ $X.t_e = Y.t_e$
X meets Y	Y met-by X		$X.t_e = Y.t_s$
X overlaps Y	Y overlapped-by X		$X.t_s < Y.t_s$ $X.t_e > Y.t_s$ $X.t_e < Y.t_e$
X during Y	Y contains X		$X.t_s > Y.t_s$ $X.t_e < Y.t_e$
X starts Y	Y started-by X		$X.t_s = Y.t_s$ $X.t_e < Y.t_e$
X finishes Y	Y finished-by X		$X.t_s > Y.t_s$ $X.t_e = Y.t_e$

Fig. 1. The thirteen possible relationships between two intervals X and Y

Relationship among time intervals can be described in different ways. **Allen's taxonomy of temporal relationships** [3] is adopted to describe the basic relationships among events and summarized in Figure 1. The relations between intervals can be expressed in terms of relations between their endpoints, we call these the *endpoint constraints*. For instance, consider the sequence $\langle E_1$=(A,5,10), E_2 =(B,8,12)$\rangle$, we have "A overlaps B" since $E_1.t_s < E_2.t_s$, $E_1.t_e > E_2.t_s$ and $E_1.t_e < E_2.t_e$.

These thirteen relationships can be used to express any relationship held between two intervals and they provide a basis for the description of temporal patterns. Some of the relations are mirror image of the other, for example, "X overlaps Y" implies the same relation as "Y is overlapped-by X". We only focus

on seven primitive temporal relations with the order of items preserved. The seven relations are shown in the shaded area in Figure 1. Let us call this set of seven temporal relations **Rel**. We obtain binary temporal predicates if we consider two events only. To express complex relation among more events in a sequence, we have the following de nitions:

De nition 1. *A* **temporal pattern** *is de ned recursively as follows:*

- *If E is a single event type in* **E**, *then E is a temporal pattern, it is also called an* **atomic pattern**.
- *If X and Y are two temporal patterns, and $rel \in$* **Rel**, *then $(X\ rel\ Y)$ is also a temporal pattern. This is a* **composite pattern**.

The **size** *of a temporal pattern is the number of atomic patterns in the pattern.*

De nition 2. *An atomic temporal pattern X has a* **mapping** *in a sequence S if we can nd an event E of type X in the sequence. We denote this mapping by $\mathcal{M}(X,S) = \{E\}$. We associate a time duration to the mapping as follows: $\mathcal{M}(X,S).t_s = E.t_s$, $\mathcal{M}(X,S).t_e = E.t_e$ We say that X* **holds** *in S.*

A composite pattern $(X\ rel\ Y)$ in which Y is an atomic pattern and $rel \in$ **Rel** *has a* **mapping** *$\mathcal{M}((X\ rel\ Y),S)$ in a sequence S if X has a mapping $\mathcal{M}(X,S)$ in S and we can nd an event $E \notin \mathcal{M}(X,S)$ of type Y in S to be mapped as $\mathcal{M}(Y,S)$ such that if we consider some imaginary event Z with start time of $\mathcal{M}(X,S).t_s$ and end time of $\mathcal{M}(X,S).t_e$, then $Z\ rel\ E$ is true.*

In this case, $\mathcal{M}(X\ rel\ Y,S) = \mathcal{M}(X,S) \cup \{E\}$. We say that the relation $(X\ rel\ Y)$ **holds** *in S. The mapping $\mathcal{M}((X\ rel\ Y),S)$ has an associated time interval given by*

$$\mathcal{M}((X\ rel\ Y),S).t_s = min\{\mathcal{M}(X,S).t_s, \mathcal{M}(Y,S).t_s\}$$
$$\mathcal{M}((X\ rel\ Y),S).t_e = \mathcal{M}(Y,S).t_e$$

In the above mapping of a composite pattern in a sequence, **union** of two time intervals takes place. For example, for the sequence shown in Figure 2(a), we obtain a composite pattern "(A overlaps B) before C" with the resultant interval being [5,18].

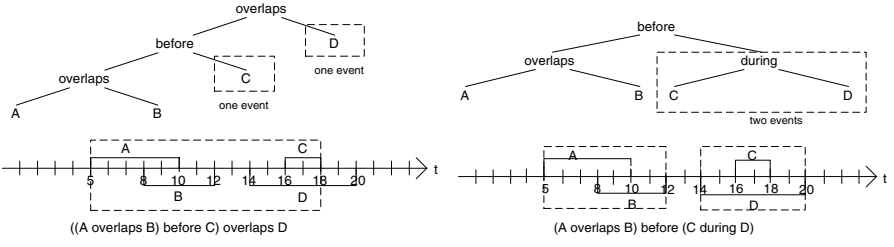


Fig. 2. Different temporal pattern representation

We can see that in Figure 2, two different composite patterns "((A overlaps B) before C) overlaps D" and "(A overlaps B) before (C during D)" hold in the sequence by considering different ways of combinations. The number of such possible descriptions is exponential in the sequence size. We restrict our interest to temporal patterns of the form $((...(A_1\ rel_1\ A_2)\ rel_2\ A_3)...\ rel_{k-1}\ A_k)$. We call these the $A1$ **temporal pattern**.

There are two main reasons for this restriction: (1) We believe that the temporal relations give some insight into causal relationships. As such, when a few events have happened, together they may become the cause of a following event. (2) Discovering all possible patterns can be computationally inhibitive and also the amount of results to the user can be overwhelming. We have implemented the mechanism to discover temporal pattern obtained by appending composite pattern of size two at a time as shown in Figure 2(b). Experimental results show that even a small extension in this way would lead to a much increase in computational cost.

Moreover, user can specify the maximum length of time interval that is of interest, known as the **window-size**, win_size. The intuition is that if some events do not happen close enough to each other, we would not be interested to find any correlation between them. If w is a given window-size, a temporal pattern P **holds within** w in a sequence S if there is a mapping $\mathcal{M}(P,S)$ such that $\mathcal{M}(P,S).t_e - \mathcal{M}(P,S).t_s \leq w$.

Definition 3. *Let A_i, $i = 1,...,k$ be a bag of k event types in* **E**, *$rel_i \in$* **Rel**, *i = 1,..,k-1, a* **k-item** *has the form $\{\{A_1, A_2, ...A_k\}, \{rel_1, rel_2, ...rel_{k-1}\}, \mathcal{P}\}$, where $\mathcal{P}$ is a temporal pattern in terms of the events types $A_1, A_2, ..., A_k$ and the relations $rel_1, ..., rel_{k-1}$, and $k \geq 1$.*

Given a window-size w, let M be a subset of the set of events in a sequence S, M **supports** an k-item $\{\{A_1, A_2, ...A_k\}, \{rel_1, rel_2, ...rel_{k-1}\}, \mathcal{P}\}$ if M is a mapping for the temporal pattern $\mathcal{P}$ and $M.t_e - M.t_s \leq w$.

The **support** of a temporal pattern $\mathcal{P}$ in a set of sequences $\mathcal{D}$ is defined as

$$support(\mathcal{P}, \mathcal{D}) = \frac{|\{M \subseteq S | S \in \mathcal{D}, M\ supports\ \mathcal{P}\}|}{|\mathcal{D}|}$$

The **support** of a k-item is the support of the temporal pattern in the k-item.

A **large k-item** is a k-item having support greater than a threshold, namely min_sup provided by users, that is, $support(\mathcal{P}, \mathcal{D}) \geq min_sup$. Our aim is to find the large k-items for all k.

3 Mining A1 Temporal Patterns

Here we propose a method for mining the frequent $A1$ temporal patterns as shown in Figure 2(a). Consider a patient database shown in Table 1. Each tuple contains a person-id, the disease contracted by the patient and the duration of the disease. The database can be used to find if some diseases are likely to cause

person-id	disease	start	end
1	A	5	10
1	B	8	12
1	D	14	20
1	C	16	18
1	B	17	22
2	A	8	14
2	B	9	15
2	D	16	22
2	C	19	21
3	A	12	20
3	B	22	25
3	D	28	32
3	C	29	31

Original database

person-id	seq-list
1	(A,5,10),(B,8,12),(D,14,20), (C,16,18),(B,17,22)
2	(A,8,14),(B,9,15), (D,16,22),(C,19,21)
3	(A,12,20),(B,22,25), (D,28,32),(C,29,31)

seq_list

item	A	B	D	C
	(1,5,10)	(1,8,12)	(1,14,20)	(1,16,18)
	(2,8,14)	(1,17,22)	(2,16,22)	(2,19,21)
	(3,12,20)	(2,9,15)	(3,28,32)	(3,29,31)
		(3,22,25)		

item_list

Table 1. Transform the database as *seq_list* and *item_list*

some other diseases and their temporal relations. It is assumed that the *min_sup* is 66% and the *win_size* is set to be 100 time units. We use a different layout of event sequence that used in nding sequential pattern [2]. Instead of transforming the original database into a list of sequences, the *seq_list*, where each sequence corresponds to one person, we use an *item_list* to represent the temporal data. Each event is associated with a list of person-id, start time and end time (*pid*, t_s, t_e). Tables 1 illustrates the differences between the two approaches. Note that each item is an atomic or composite pattern and the t_s and t_e in the *item_list* indicates an time interval for this pattern. Similar idea of transforming the database into the form which we called *item_list* is used in [10].

A basic strategic similar to the Apriori-gen approach [1] is used. With *seq_list*, we need to scan the database once in each iteration. The *item_list* approach avoids this problem since it enables us to count support by direct composition of the lists. The size of these lists and number of candidates shrink as the sequence length increases, which facilitates fast computation. This motivates us to choose *item_list* format to store the large k-items for efficient support counting.

The general structure of the method for the mining process is mainly divided into two phases: (1) **Candidate Generation** and (2) **Large k-items Generation**. It takes multiple passes over the data. In each pass, we start with a seed set of large items which is found in the previous pass. We use the seed set for generating new potentially large items, called candidates, C_k, by adding one atomic item to elements in the seed set. In the second phase, for each candidate, we examine the L_{k-1} and L_1 *item_lists* and determine the temporal relations between the composite pattern and atomic pattern that have sufficient supports. We then generate new large k-items and obtain the L_k *item_list* by merging the composite and atomic patterns with the temporal relation. The algorithm

terminates when we cannot find any large k-items after the end of the current pass.

Candidate Generation:

The logical form of the candidates is shown in an example in Table 2. The candidate generation to obtain C_k from L_{k-1} is done by adding one large 1-item each time. We generate the candidates by combining the events in atomic patterns with those in composite patterns.

item	count	item	count	item	count
A overlaps B	2	A before B	2	C during D	3
(1,5,12)		(1,5,22)		(1,14,20)	
(2,8,15)		(3,12,25)		(2,16,22)	
				(3,28,32)	

Partial large 2-items

item 1	item 2
A overlaps B	C
A overlaps B	D
A before B	C
A before B	D
⋮	⋮

The 3-candidates

Table 2. Partial large 2-items and the 3-candidates

In each pass, we generate C_k from L_{k-1}. We prune out some irrelevant candidates as follows: let b_i be the $k-1$-item with $((..(a_{i1}rel_{r1}a_{i2})...rel_{rk-2}a_{ik-1})$ where $rel_{ri} \in$ Rel if $(a_{i1}rel_{ij}a_j) \in L_2$ or $(a_{ik-1}rel_{ij}a_j) \in L_2$ for any rel_{ij} then take the candidate element $\{b_i, a_j\}$. For instance, for the 2-item with a pattern of "C during D", we aim to find any temporal relation between the 2-item and event A. In the previous pass, we have found that no pattern of the form "C *rel* A" or "D *rel* A" is large, we therefore exclude the possibility of having the candidate of "(C during D)" with "A" in the generation of large 3-item.

Large k-Items Generation:

This phase is further divided into two subphases. They are the support counting phase and generation of large k-items. First, we need to find the supports for the candidates generated. We determine the number of sequences that support each temporal relation between the composite patterns in each candidate. We compare the endpoint values of elements in L_{k-1} and L_1 and determine if any temporal relation holds between the composite and atomic patterns. Large k-items are formed if their support is greater than the threshold given.

To facilitate efficient counting, we use a hash tree to store L_1 and also the relevant part of the 1-*item_list* and a hash tree to store L_{k-1} and part of the $(k-1)$-*item_list*. We use the value of the event as a key for hashing in the hash tree for L_1. For the hash tree for composite patterns, we use the values of all the events included and the temporal relations together to form a key by simple concatenation. The leaf nodes of the hash tree corresponds to some large k-item I and it also records the mappings for the pattern in I. The mappings are stored in the form of *item_list*, with the person-ids, start times and end times. For each candidate, we use the composite pattern and event as search keys and find from the hash trees the corresponding L_{k-1} and L_1 items.

During the search in the hash tree, hashing is done in the internal nodes until we reach the leaf nodes where we perform exact matching for the composite or atomic pattern. We consider a pattern P for a large $(k-1)$-item and a pattern P' for a large 1-item. We can identify any temporal relation that holds between a mapping for P and a mapping in P', since the start times and end times are recorded in the corresponding hash trees. If some composite pattern is found, the count for the candidate with respect to the speci c temporal pattern is increased by 1. The counts are kept as auxiliary information to the table for the candidate items. There are seven counts for each candidate, one for each of the temporal relationships.

The second subphase is to form large k-items. Table 2 shows a partial set of the large 2-items. After identifying any temporal relation between the items in C_k, we generate L_k from L_{k-1} and L_1 with the corresponding temporal relation. Each new item in L_k is a composite pattern and is used for the next pass. The resultant interval is obtained from the union of two intervals of $(k-1)$-item and 1-item. For instance, as shown in Table 2, the start time and end time of the mapping of composite pattern "A overlaps B" are {[5,12], [8,15]}.

4 Mining Other Temporal Patterns

In the previous section, in mining the A1 temporal pattern, we generate large k-items by adding one atomic pattern in L_1 at a time. Here we consider a slightly more complex form of temporal patterns which we call A2 temporal pattern. The A2 temporal pattern is de ned recursively as the following: (1) a temporal pattern of size 2 is an A2 temporal pattern. E.g. "A before B". (2) if X is an A2 temporal pattern, and Y is a temporal pattern of size 2, then $(X\ rel\ Y)$ where $rel \in$ **Rel** is also an A2 temporal pattern. Example of such a composition is shown in Figure 2(b). The patterns we generate are in even number of events, i.e. $2k$-items. We therefore focus on temporal relations among events by adding one 2-item each time. We shall refer to the method in Section 3 as the **rst method** and the method described in this section as the **second method**. The second method works similarly as the previous method except for the candidate generation phase and the formation of large $2k$-items.

Candidate Generation:

In the formation of C_2, the process is the same as before. Next we start to generate C_{2k}, where $k \geq 2$, we examine L_{2k-2} and L_2 and use compositions of the elements in the two *item_list* of L_{2k-2} and L_2. When we prune any irrelevant candidates in this phase, we need to consider two newly added atomic patterns this time, say a_{j1} and a_{j2}, where $a_{j1} rel_j a_{j2} \in L_2$. The two added items can be combined with a composite pattern, say b_i, where $b_i \in L_{2k-2}$ if both of the following conditions hold:

1. there is a relation between the leftmost atomic pattern of b_i and at least one of a_{j1} and a_{j2}.
2. there is a relation between the rightmost atomic pattern of b_i and a_{j1} or a_{j2}.

Large $2k$-Items Generation:
We also divide this into two phases namely support counting and the generation of large $2k$-items. In general, the second method works in the same manner as the first method in that we generate incrementally larger $2k$-items by combining the two composite patterns of a $2k$-candidate. The difference is that we shall use the *item_list* of L_{2k-2} and L_2.

5 Performance Evaluation

To evaluate the performance of the proposed methods over a large range of data, we conducted several experiments on an UltraSparc 5/270 workstation with 520MB of main memory. The two methods in Sections 3 and 4 are written in C. We consider synthetic data in an application domain of a medical database same as that of the given example. For each person we record a sequence of clinical records stating the different diseases contracted.

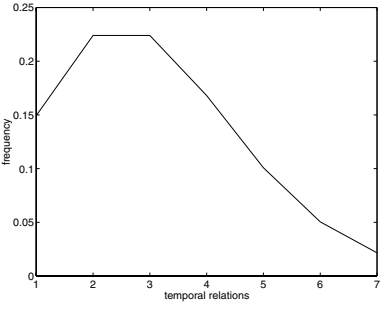


Fig. 3. Distribution of temporal relations between events

The synthetic data generation program took four parameters. They are number of sequences ($|D|$), average number of events per large item ($|P|$), number of maximal potentially large item, (N_S), and number of event types (N). The data generation model was based on the one used for mining sequential pattern [2] with some modification to model the patient database. We generated each large sequence by first picking a number of events from a Poisson distribution with mean μ equal to $|P|$ and we chose the event types randomly. We picked a temporal relation between events and formed a pattern. Temporal relations are chosen from the set **Rel**, according to a distribution shown in Figure 3. We used the values of {1,2,3,4,5,6,7} to represent *starts, overlaps, before, during, finishes, meets, equal* respectively. The distribution was determined arbitrarily by our intuitive expectation of the likeliness of the relations. Each person was then assigned a potentially large sequence. The time interval of each event followed an exponential distribution with mean μ equal to 5. For each sequence, the time where the first event took place was chosen randomly within the time

interval [0,500] of time units. The following events of the item then followed the temporal relation held between events. We generated the dataset by setting N_S=2000, N=1000, $|D|$=10K and $|P|$=5 with 1MB of data. We studied the effect of different values of *min_sup*, *win_size*, number of sequences and events per sequence for the two methods.

First, we studied the effect of minimum support on the processing time. We used 7 values of minimum support (*min_sup*) as shown in Figure 4(a), and 100 time units for window size (*win_size*) for the test. It shows the execution time decreases when the minimum support increases for both methods. As the support threshold increases, less large items are generated and hence the size of the candidate set in each iteration decreases dramatically. Thus less time is required for support counting and hash tree searching of large items. Comparing the two methods, the second method needs much more time than that of the first method. This is due to large amount of computation time in pruning the candidates as the addition of two atomic patterns are considered instead of one.

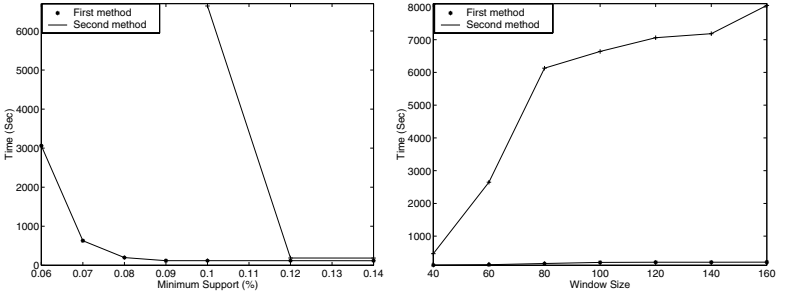


Fig. 4. Variation on minimum support and window size

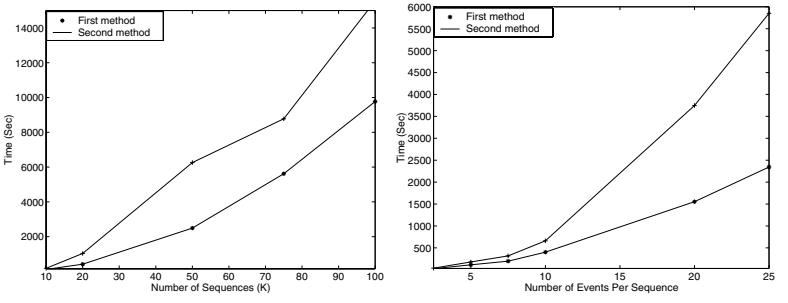


Fig. 5. Scale-up: Number of sequences and Number of events per sequence

We then studied the effect of window size on the processing time. We chose the values of min_sup being 0.0008 for the first method and 0.001 for the second method. In Figure 4(b), we can see that when the window size increases, the execution time increases for both methods. This is because more sequences are included and the time for support counting phase increases. Also the number of iterations increases and which also requires a longer execution time.

Our next target is to consider the scale-up effects. Figure 5(a) shows how both methods scale up as the number of sequences is increased ten-fold, ranging from 10K to 100K and with $min_sup = 0.0025$ for both methods. The execution time for both methods increase with increasing number of sequences.

Finally we studied the scale-up as we increase the average number of events per sequence. The number of sequences used is 10K and kept constant. We vary the average number of events per sequence from 2.5 to 25. Figure 5(b) shows the scalability results of the two methods. We set $min_sup = 0.0025$ for both methods. From the figure, the execution time increases rapidly with the increasing number of events per sequence.

In conclusion, we propose methods for discovering interesting temporal patterns for interval-based events. A set of experiments has been conducted to demonstrate the overall performance of the methods. From experiments, we find that the computation time required for the first pattern is much more acceptable.

References

1. R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proceedings of the 20th VLDB Conference*, pages 487–499, 1994.
2. R. Agrawal and R. Srikant. Mining sequential patterns. In *11th International Conf. on Data Engineering*, pages 3–14, March 1995.
3. J. F. Allen. Maintaining knowledge about temporal intervals. In *Communications of the ACM 26(11)*, pages 832–843, 1983.
4. X. Chen and I. Petrounias. An architecture for temporal data mining. In *IEE Colloqium on Knowledge Discovery and Data Mining*, pages 8/1–8/4, 1998.
5. H. Mannila and H. Toivonen. Discovering generalized episodes using minimal occurrences. In *2nd International Conf. on Knowledge Discovery and Data Mining*, pages 146–151, August 1996.
6. H. Mannila, H. Toivonen, and A. I. Verkamo. Discovering frequent episodes in sequences. In *1st International Conf. on Knowledge Discovery and Data Mining*, pages 210–215, August 1995.
7. B. Padmanabhan and A. Tuzhilin. Pattern discovery in temporal databases: A temporal logic approach. In *2nd International Conf. on Knowledge Discovery and Data Mining*, pages 351–354, August 1996.
8. M. H. Saraee and B. Theodoulidis. Knowledge discovery in temporal databases. In *IEE Colloquium on Digest No. 1995/021(A)*, pages 1/1–1/4, 1995.
9. A. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, and R. Snodgrass. *Temporal Databases: Theory, Design, and Implementation.* Benjamin/Cummings, 1993.
10. M. J. Zaki. Fast mining of sequential patterns in very large databases. Technical report, Technical Report 668 of the Department of Computer Science, University of Rochester, Nov 1997.

An Integrated Query and Mining System for Temporal Association Rules

Xiaodong Chen[1] & Ilias Petrounias[2]

[1] Department of Computing & Mathematics, Manchester Metropolitan University
Manchester M1 5GD, U.K.
e-mail: x.chen@doc.mmu.ac.uk
[2] Department of Computation, UMIST, PO Box 88, Manchester M60 1QD, U.K.
e-mail: ilias@co.umist.ac.uk

Abstract. In real world the knowledge used for aiding decision-making is always time varying. Most existing data mining approaches assume that discovered knowledge is valid indefinitely. Temporal features of the knowledge are not taken into account in mining models or processes. As a consequence, people who expect to use the discovered knowledge may not know when it became valid or whether it is still valid. This limits the usability of discovered knowledge. In this paper, temporal features are considered as important components of association rules for better decision-making. The concept of temporal association rules is formally defined and the problems of mining these rules are addressed. These include identification of valid time periods and identification of periodicities of an association rule, and mining of association rules with a specific temporal feature. A system has been designed and implemented for supporting the iterative process of mining temporal association rules, along with an interactive query and mining interface with an SQL-like mining language.

1. Introduction

Data mining is becoming an important tool for decision-making. It can be used for the extraction of non-trivial, previously hidden patterns from databases. Over the last few years, many forms of such patterns have been identified in various application domains. A typical one is the association rule [1]. Consider a supermarket database, where the set of items purchased by a customer is stored as a transaction. An example of an association rule is: "60% of transactions that contain bread and butter also contain milk; 30% of all transactions contain all these three items." 60% is called the confidence of the rule and 30% the support. The meaning of this is that customers who purchase bread and butter also buy milk. In all but the most trivial applications, however, the knowledge is time varying. Knowing when something became valid, or how often in time a pattern occurs, can provide more valuable information to individuals and organisations. For example, in retail applications, people may be more interested in the association "customers who buy bread and butter also buy milk *during the summer*" rather than the much simpler one "customers who buy bread and butter also buy milk". In this paper, temporal features are considered as important

Y. Kambayashi, M. Mohania, and A M. Tjoa (Eds.): DaWaK 2000, LNCS 1874, pp. 327-336, 2000.
© Springer-Verlag Berlin Heidelberg 2000

components of association rules. The concept of temporal association rules is formally defined and a methodology for mining them is suggested. Temporal association rules can be extracted by alternatively performing different forms of restricted mining tasks, such as identification of valid time periods and identification of periodicities of an association rule, and mining of association rules with a specific temporal feature. To support the interactive and iterative process of mining temporal association rules, an integrated query and mining system (IQMS) has been designed and implemented, with an SQL-like mining language TML (Temporal Mining Language) for describing the mining tasks.

The rest of the paper is organised as follows. Section 2 describes related work. Section 3 introduces temporal association rules and states the mining tasks for them. Section 4 presents a temporal mining support system. Experimental results on the number of discovered rules are finally discussed in section 5, followed by conclusions.

2. Related Work

The problem of finding association rules was introduced in [1]. Given a set of transactions, where each transaction is a set of items, an association rule between two sets of items, X and Y, is an expression of the form $X \Rightarrow Y$, indicating that the presence of X in a transaction will also imply the presence of Y in the same transaction. Temporal issues were not considered in the original problem modelling, where the discovered association rules were assumed to hold in the universal domain of time. However, transactions in most applications are usually stamped with time values in the database. Temporal issues of association rules have recently been addressed in [3], [7] and [8]. The major concern in [3] is the discovery of association rules for which the valid period and the periodicity are known. The valid period, such as "starting from September 1995 and ending by August 1998", shows the absolute time interval during which an association is valid, while the periodicity, such as "every weekend", conveys when and how often an association is repeated. Both the valid period and the periodicity are specified by calendar time expressions. The cyclicity of association rules was similarly discussed in [7], where a cycle of an association rule was defined as a tuple (l, o) such that the rule holds in every l^{th} time unit starting with time unit t_o. In [8] the concept of calendric association rules is defined, where the association rule is combined with a calendar, such as "all working days in 1998", that is basically a set of time intervals and is described by a calendar algebra.

3. Discovery of Temporal Association Rules: Task Description

3.1 Temporal Association Rules

Let $\boldsymbol{I}=\{i_1, i_2, \ldots, i_m\}$ be a set of literals which are called items. A set of items $X \subset \boldsymbol{I}$ is called an itemset. Let $\boldsymbol{D}$ be a set of time-stamped transactions over the time domain $\boldsymbol{T}$ which is a totally ordered set of time instants (or chronons). Each time-stamped transaction S is a triplet <tid, itemset, timestamp>, where S.tid is the transaction

identifier, S.itemset is a set of items such that S.itemset $\subseteq \boldsymbol{I}$, and S.timestamp is an instant (or a chronon) which the transaction S is stamped with, such that S.timestamp $\in \boldsymbol{T}$. A transaction S contains an itemset X if X $\subseteq$ S.itemset. In the discrete and linear model of time, a time interval is the time between two instants, basically represented by a set of contiguous instants. In general, any temporal feature TF, whatever it is, a valid period, a periodicity, or a specific calendar, represents a set of time intervals, which is called its interpretation and is denoted by $\Phi(TF) = \{P_1, P_2, ..., P_n\}$, where P_i is a time interval. For example, the interpretation of a periodic time PT, $\Phi(PT)$, is a set of periodic intervals in cycles. The interpretation of a valid time period VI, $\Phi(VI)$, consists of only a contiguous time interval. The interpretation, $\Phi(CAL)$, of a specific calendar CAL, is a set of all relevant intervals in this specific calendar.

Definition 3.1: Given a set of time-stamped transactions, a *temporal association rule* is a pair <AR, TF>, where AR is an implication of the form X⇒Y and TF is a temporal feature that AR possesses. It expresses that during each interval P in $\Phi(TF)$, the presence of X in a transaction implies the presence of Y in the same transaction. The frequency of temporal association rules is defined as follows:

- AR has confidence *c* % during interval P_i, $P_i \in \Phi(TF)$, if at least *c* % of transactions in $\boldsymbol{D}(P_i)$ that contain X also contain Y.
- AR has support *s* % during interval P_i, $P_i \in \Phi(TF)$, if at least *s* % of transactions in $\boldsymbol{D}(P_i)$ contain X∪Y.
- AR possesses the temporal feature TF with the frequency *f* % in the transaction set $\boldsymbol{D}$ (or saying the database $\boldsymbol{D}$) if it has minimum confidence min_*c* % and minimum support min_*s* % during at least *f* % of intervals in $\Phi(TF)$.

Here, $\boldsymbol{D}(P_i)$ denotes a subset of $\boldsymbol{D}$, which contains all transactions with timestamps belonging to P_i. In the above definition, the notion of frequency is introduced for measuring the proportion of the intervals, during which AR satisfies minimum support and minimum confidence, to the intervals in $\Phi(TF)$. It is required that the frequency of any temporal association rule < AR, TF > should not be smaller than the user-specified minimum frequency which is a fraction within [0,1]. In the case that $\Phi(TF)$ just includes a single interval, the frequency will be omitted since the meaningful minimum frequency must be 1 and AR must have minimum support and minimum confidence during this single interval. Temporal features can be represented by calendar time expressions [4]. For example, the expression "*years* · *months* (6:8)" represents "every summer". Depending on the interpretation of the temporal feature TF, a temporal association rule <AR, TF> can be referred to as:

- a *universal association rule* if $\Phi(TF) = \{ \boldsymbol{T} \}$, where $\boldsymbol{T}$ represents the time domain;
- an *interval association rule* if $\Phi(TF) = \{ itvl \}$, where itvl $\subset \boldsymbol{T}$ is a specific time interval (e.g., a time period from July 11, 1998 to September 6, 1998);
- a *periodic association rule* if $\Phi(TF) = \{ p_1, p_2, ..., p_n \}$, where $p_i \subset \boldsymbol{T}$ is a periodic interval in cycles (e.g., a weekend);
- a *calendric association rule* if $\Phi(TF) = \{ cal_1, cal_2, ..., cal_m \}$, where $cal_i \subset \boldsymbol{T}$ is a calendric interval in a specific calendar (e.g., the first term in the 1998/1999 academic year).

3.2 Mining Tasks for Temporal Association Rules

The mining of temporal association rules has a two-dimensional solution space consisting of rules and temporal features. From a practical point of view, it is too expensive to find all hidden temporal association rules from large databases without any given information. The methodology adopted in this paper is simplifying the mining problem into restricted mining tasks, seeking different techniques for these restricted tasks, and alternatively using these techniques in an interactive and iterative mining process with the support of a data mining system. These are identified below.

3.2.1 Discovery of Longest Intervals in Association Rules

In many applications, people might just be interested in specific associations of some items in the database, but have no any idea about when and/or how often these associations hold. These specific associations could be speculated by domain experts or may have been found in earlier efforts during the iterative data mining process. One temporal feature that is of interest is the valid time period of such associations. Given a time-stamped database and a known association, one of our interests is to find all possible time intervals over the time domain during which this association holds. Each of those time intervals will be composed of a totally ordered set of granules [2]. Here, a granule is a minimum interval of some fixed duration, by which the length of any time interval is measured. The granularity (the size of a granule) may vary with different applications and users' interests. The interval granularity is normally greater than the granularity of instants with which the data are stamped. For example, in a supermarket database, a transaction may be stamped by "minute" (i.e., the instant granularity is "minute"), but an association may be looked for over the period of a day or several days (i.e., the interval granularity is "day"). More often than not people are just interested in the intervals, during which an association holds and the duration of which is long enough. In terms of different applications, a minimum length of the expected intervals can be specified by the users. We define the interestingness of time intervals as follows:

Definition 3.2: A time interval over the time domain is *interesting with respect to* an association rule (AR) if it is longer than the user-specified *minimum length* and AR holds (satisfies the minimum support and confidence) during this interval and any sub-interval of this interval that is also longer than the user-specified minimum length.

Given a set of time-stamped transactions (***D***) over time domain (***T***) with an interval granularity (*GC*), a minimum support (*min_supp*), a minimum confidence (*min_conf*), and a minimum interval length (*min_ilen*), the task of mining valid time periods is to find *all the longest interesting intervals* with respect to a given association *AR*.

3.2.2 Discovery of Longest Periodicities in Association Rules

Another interesting temporal feature is a set of regular intervals in cycles, during each of which an association exists. In this paper, a set of periodic intervals in cycles is called a periodic time, which is composed of three essential features as follows: *Cyclicity* (a periodic time is always based on a fixed-length cycle, such as a month, a week, etc.), *Granularity* (each periodic interval in cycles consists of a series of granules of some fixed duration), *Interval Range* (all the regular intervals in a

periodic time are located in the same position in their corresponding cycles). Therefore, a periodic time can be represented by a triplet <Cyclicity, Granularity, Range>, which is called *an essential representation* of the periodic time. For example, <Year, Month, [10:12]>, <Week, Day, [2:6]> and <Month, Day, [1:10]> represent "last quarter of every year", "working days of each week", and "the first ten days of every month", respectively.

Given a periodic time PT = < CY, GR, [x:y] >, its interpretation Φ(PT) = {P_1, P_2, ..., P_j, ...} is regarded as a set of periodic intervals consisting of the x-th to y-th granules of GR, in all the cycles of CY. We define the interestingness of a periodic time with respect to an association on the basis of the concept of interesting intervals.

Definition 3.3: A periodic time PT=<CY, GR, RR> is *interesting with respect to* an association rule (AR) if the ratio of its periodic intervals in cycles that is interesting with respect to AR is not less than a user-specified *minimum frequency*. PT is a longest interesting periodic time with respect to AR if: 1) PT is interesting with respect to AR, and 2) not $\exists$ PT' = < CY, GR, RR' > such that RR' $\supset$ RR and PT' is interesting with respect to AR.

Given a set of time-stamped transactions (***D***) over a time domain (***T***), a minimum support (*min_supp*), a minimum confidence (*min_conf*), a minimum frequency (*min_freq*), a minimum interval length (*min_ilen*), as well as the cycle of interest(*CY*) and granularity (*GR*), *the task of mining the periodicities of an association (AR) is* to find all possible periodic times < *CY*, *GR*, *RR* >, which are longest with respect to the association *AR*. Here, the range RR is expected to be discovered.

3.2.3 Discovery of Association Rules with Temporal Features

The mining of associations with specific temporal features that reflect the crucial time periods in the business might also be of interest to some applications. In this case, only the behaviour over those periods is crucial. Those interesting periods are characterised by temporal features, such as a single time period (e.g., from July 1996 to January 1998), a periodic time (e.g., every Friday), or a specific calendar (e.g., payment days in 1998). Given a set ***D*** of time-stamped transactions over a time domain ***T*** and user-specified thresholds (minimum support *min_s %*, confidence *min_c %* and minimum frequency *min_f %*), *the task of mining association rules with a given temporal feature TF* is to discover all possible associations of the form X$\Rightarrow$Y in ***D*** that possess the temporal feature *TF* with the minimum frequency *min_f %*.

4. An Integrated Query and Mining System

The search techniques and algorithms for the above mining tasks for temporal association rules have been presented in ([3], [4]) and implemented in an integrated query and mining system (IQMS) presented here. The IQMS system supplies mining users with an interactive query and mining interface (IQMI) and an SQL-like Temporal Mining Language (TML).

To support the iterative process of the discovery of temporal association rules, the IQMS system has been integrated with the functions of both data query and data mining. By using this mining system, people can make either an ad hoc query for interesting data or an ad hoc mining for interesting knowledge, with an interactive query and mining interface (IQMI) (see Figure 4.1). This interface connects a user to

any Oracle database and prompts the user to submit any SQL or TML statement. If it is a query, the query result will be displayed. If it is a mining task, the mined rules are displayed. The system has been implemented with Oracle Pro*C/C++ under a Sun SPARCstation 20 running SunOS 5.5.

An SQL-like data mining language [6] is one of the important tools to support the interactive mining process. As the kernel of the IQMS system, the TML language supplies users with the ability to express various mining tasks for temporal association rules discussed in section 3.2. The structure of TML is described as follows:

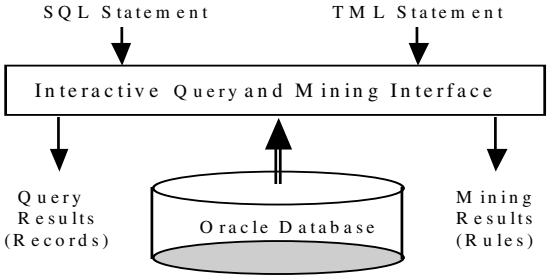


Figure 4.1 Interactive Query and Mining Interface

```
1)      Mine Association_Rule ( <rule-specification> )
2)           [ With Periodicity ( <periodic-specification> ) ]
3)           [ Within Interval ( <interval-specification> ) ]
4)           [ Having Thresholds <threshold-specification> ]
5)      In   <query-block>
```

A mining task in TML consists of the mining target part (lines 1-4) and the data query block. The keyword Association_Rule is a descriptor, followed by the rule specification. The rule specification is either a keyword "ALL" or a specific rule. The former indicates that all of potential rules should be found from the database and the latter expresses that temporal features of this rules are to be extracted from the database. The periodicity of rules is specified by the periodic specification, which is either an instantiated periodic expression (e.g., Month*Day(1:5)) or a uninstantiated one (e.g., Week*Day(ALL)). The former shows that only rules with this periodicity are expected, while the latter expresses that all possible periodicities of a specific rule are expected. The interval specification can be used for describing the valid period of rules, where the interval specification is either an instantiated interval expression (e.g., After Year(1998)) or a uninstantiated one (e.g., Day(ALL)). The former indicates the specific time period that users are interested in, while the latter expresses that all contiguous time intervals during which rules (periodic or non-periodic) may exist are to be extracted. Thresholds relevant to different tasks can be stated in the Having-Threshold clause. The data relevant to the data mining task can be stated in the query block. Consider a Sales database containing two relations Items and Purchase as shown in Figure 4.2. Three examples of mining tasks in TML are given below.

Example 4.1: Mining Longest Intervals of an Association Rule

```
Mine Association_Rule ("Coca Cola" ⇒ "Tesco Brown Bread")
    Within Interval ( day(ALL) )
    Having Thresholds support = 0.3, confidence = 0.75, interval_len = 6
```

In Select trans_no:Tid,Set(item_name):ItemSet,trans_time: TimeStamp
From purchase, items
Where purchase.item_no = items.item_no
Group By purchase.trans_no ;

This example describes a task of mining all longest intervals of an association rule with the thresholds of support and confidence being 0.3 and 0.75, respectively. The interval granularity is specified as 'day'. The threshold expression "interval_len = 6" states that the length of all discovered intervals must be at least 6 days long. In this example, the "Select-From-Where-Group" part expresses a basic data query demand. The result of the data query is a nested relation, forming the data set relevant to this mining task. "Trans-no" and "Set(item-name)" are considered as "Tid" and "Itemset" in the mining model for temporal association rules, respectively, and "trans-time" in the database is chosen as the time stamp.

Items

item-no	item-name	brand	category	retail-price

Purchase

trans-no	trans-time	customer	item-no	qty	amount

Figure 4.2 Relation Schemes in Sales Database

Example 4.2: Mining Periodicities of An Association Rule

Mine Association_Rule ("Coca Cola", "Flora Butter" ⇒ "Tesco White Bread")
With Periodicity (week*day(ALL))
Within Interval (starts_from year(1990) and finishes_by year(1998))
Having thresholds support = 0.3, confidence = 0.75, frequency = 0.8, interval_len = 2
In Select trans_no: Tid, Set(item_name): ItemSet, trans_time: TimeStamp
From purchase, items
Where purchase.item_no = items.item_no
Group By purchase.trans_no ;

The mining task in this example is to look for all the periodicities of a specific association rule between 1990 and 1998, with the cyclicity being 'week' and the granularity being 'day'. The minimum interval length of two days. The thresholds of support, confidence and frequency are 0.3, 0.75, and 0.8, respectively.

Example 4.3: Mining Association Rules With a Periodicity

Mine Association_Rule (*ALL*)
With *Periodicity(Years·Months(6:8))*
Within Interval (*starts_from Years(1990)·Months(7)*)
Having Thresholds support = 0.6, confidence = 0.75, frequency = 0.8
In Select *trans-no: TID, Set(item-name): ItemSet, trans-time:TimeStamp*
From purchase, items
Where purchase.item-no = items.item-no and items.retail-price >= 10
Group By purchase.trans-no Having Count(*) > 3

The above is an example of a mining task for finding all periodic association rules that convey purchase patterns every summer (assuming that summer starts from the seventh month of each year) since July 1990, with the thresholds of support, confidence and frequency being 0.6, 0.75 and 0.8, respectively. In this example, we are only concerned with items whose retail prices are not less than £10 and transactions in which the number of purchased items is greater than 3.

By using this system, data miners can explore knowledge from databases in an iterative process as shown in Figure 4.3. With the query function, the data in the database can firstly be analysed, according to the business requirements, in order to get some useful information (e.g., summary information about the data) for designing mining tasks. Then, the mining task, expressed in the TML language, can be designed by the data miner and fulfilled by the ad hoc mining function of the system. The mining results need to be further analysed to judge if the expected knowledge has been found or if the mining task should be adjusted to make another mining effort. The adjustment could be, for example, the change of the values of thresholds in the mining target part, or a modification on the restriction of selected data in the data query part.

A submitted TML statement (a mining task) will be processed in the following steps:

Task Analysis: The statement is syntactically and semantically analysed. The outcomes of this step are: i) a query demand is generated for preparing the data relevant to the mining task and is represented in an SQL-like intermediate language, and ii) an internal mining problem representation consisting of mining target (e.g., rules, longest intervals, or periodicities) and thresholds (i.e., minimum support, confidence, frequency and interval length) is generated. The generated query and extracted mining target representation will be used in the next two steps successively.

Data Preparation: The data relevant to the mining task are prepared (according to the query demand generated) in this step. These data are selected from the database and are organised in the format required by the mining algorithms. To reduce the redundant data access to operational databases, an internal *data cache* is used. The data cache consists of a fixed number of temporary tables, storing selected data sets relevant to the most recently performed mining tasks. The corresponding information about each data set in the cache is organised by a *cache manager*.

Rule Search: The search for possible rules from the prepared data is performed in this step, according to the mining target and thresholds. In terms of the mining target, the system will execute the appropriate search algorithm. These algorithms have been presented in [3], [4].

Current commercial DBMS products support only a few limited operations on date and time types. Some specific time mechanisms are required for the processing of temporal aspects in the mining tasks. In this system, a number of time-related functions have been implemented.

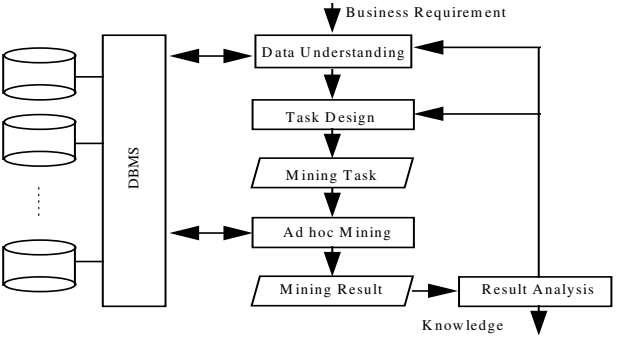


Figure 4.3 IQMI-Based Mining Process

5. Experimental Results

Synthetic data have been widely used for the evaluation of the mining of association rules ([5], [7]). Three datasets that mimic the transactions within one year in a retailing application have been generated and used for evaluating the approach presented in this paper. Each transaction in the datasets is stamped with the time instant at which it occurs. We assume that the numbers of transactions in different granules (each day) within a cyclic period (each month) are binomially distributed and the number of transactions occurring during each cyclic period is picked from a Poisson distribution with a given mean value. The average size of transactions is set as 10 and the average number of transactions during each granule (a day) is set as 100. All the datasets are stored in the Oracle 7.3.2 system. The performance analysis of the algorithms implemented in the system is out of range of this paper. Here, the discussion will only focus on the number of discovered rules. Figure 5.1 depicts the number of discovered universal rules by not considering time issues, and the number of discovered temporal rules with the periodicity <*Month, Day,* [13:17]>, which satisfy the minimum support (0.05) and confidence (0.6) and different minimum frequencies from 0.1 to 1.0. These results shows that a number of rules that may occur during a series of time periods of interest could be missed if only the approach for mining universal association rules is considered. For example, only 17 universal rules can be discovered from dataset 2 if temporal features are not taken into account. However, there are 39 rules that satisfy the support and confidence thresholds during the time periods between the 13th day and the 17th day within at least three months (see the column marked 0.3 in Figure 5.1) . If the users are more interested in the rules occurring during this specific period (between 13th and 17th), the approach for mining temporal association rules should obviously be adopted. Moreover, the techniques for mining temporal features of rules can further be used if the actual time periods during which the rules occur are expected to be identified by data miners. In addition, it is also shown in Figure 5.1 that the value of minimum frequency is one of the most important factors that decide the number of discovered rules. It is important to choose an appropriate threshold according to application requirements, in order to discover all potential useful rules. Consider dataset one, for example, when the minimum frequency is set to be greater than 0.6, the potential rules discovered will be rare, while if the minimum frequency is too small (say, 0.1), the number of potential rules discovered will be very large (see the first column on the left in Figure 5.1).

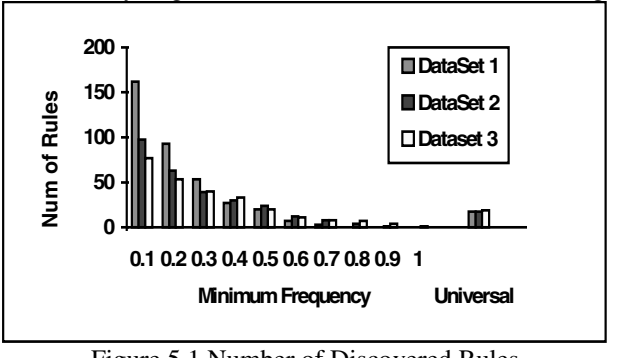


Figure 5.1 Number of Discovered Rules

6. Conclusions

The concentration of this paper is on the temporal aspects of association rules. Three forms of mining tasks for temporal association rules were addressed. By alternatively fulfilling these mining tasks, temporal association rules can be extracted from databases in an iterative and interactive process. This process is supported by the integrated query and mining system with an interactive query and mining interface (IQMI) and an SQL-like temporal mining language (TML). This paper introduced the IQMI interface, described the TML language, presented the IQMI-based mining process, and discussed the implementation issues. The results of experiments with synthetic datasets show that many time-related association rules that would have been missed with traditional approaches can be discovered with the approach presented in this paper. The significance of the integrated system has been shown in three aspects in the dynamic data mining efforts. Firstly, the *data selection and sampling* for different data mining tasks are easy to achieve with the query function that is integrated in the system. Secondly, *ad-hoc mining* for different application requirements is possible to fulfil with the mining language supported by the system. Finally, all data mining activities can be undertaken in a more flexible data mining process based on IQMI.

References

1. R. Agrawal, T. Imielinski, and A. Swami. Mining Associations between Sets of Items in Massive Databases, In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pp.207-216, Washington D.C., USA (1993).
2. C. Bettini, C. E. Dyreson, W. S. Evans, R. T. Snodgrass, and X. S. Wang. A Glossary of Time Granularity Concepts, In *Temporal Databases: Research and Practice*, Springer-Verlag. 1998
3. X. Chen, I. Petrounias, and H. Heathfield. Discovering Temporal Association Rules in Temporal Databases, In *Proceedings of International Workshop on Issues and Applications of Database Technology*, pp.312-319, Berlin, Germany (1998).
4. X. Chen and I. Petrounias, Mining Temporal Features in Association Rules, In *Proceedings of* of the 3rd European Conference on Principles and Practice of Knowledge Discovery in Databases, pp.295-300, Prague, Czech Republic (1999).
5. T. Fukuda, Y. Morimoto, S. Morishita, and T. Tokuyama. Mining Optimized Association Rules for Numeric Attributes, In *Proceedings of the 15th ACM SIGACT-AIGMOD-SIGART symposium on Principles of Database Systems*, pp.182-191, Montreal, Canada (1996).
6. R. Meo, G. Psaila, and S. Ceri. A New SQL Like Operator for Mining Association Rule, In *Proceedings of the 22nd International Conference on Very Large Databases*, pp.122-133, Bombay, India (1996).
7. B. Ozden, S. Ramaswamy, and A. Silberschatz. Cyclic Association Rules, In *Proc. of the 14th International Conference on Data Engineering*, pp.412-421, Orlando, Florida, USA (1998).
8. Ramaswamy, S., Mahajan, S., and Silberschatz, A., *On the Discovery of Interesting Patterns in Association Rules*, Proceedings of 24th VLDB Conference, New York, pp.368-379, 1998.

Mining Changes for Real-Life Applications

Bing Liu, Wynne Hsu, Heng-Siew Han and Yiyuan Xia

School of Computing
National University of Singapore
3 Science Drive 2
Singapore 117543
{liub, whsu, xiayy}@comp.nus.edu.sg

Abstract. Much of the data mining research has been focused on devising techniques to build accurate models and to discover rules from databases. Relatively little attention has been paid to mining changes in databases collected over time. For businesses, knowing what is changing and how it has changed is of crucial importance because it allows businesses to provide the right products and services to suit the changing market needs. If undesirable changes are detected, remedial measures need to be implemented to stop or to delay such changes. In many applications, mining for changes can be more important than producing accurate models for prediction. A model, no matter how accurate, can only predict based on patterns mined in the old data. That is, a model requires a stable environment, otherwise it will cease to be accurate. However, in many business situations, constant human intervention (i.e., actions) to the environment is a fact of life. In such an environment, building a predictive model is of limited use. Change mining becomes important for understanding the behaviors of customers. In this paper, we study change mining in the contexts of decision tree classification for real-life applications.

1. Introduction

The world around us changes constantly. Knowing and adapting to changes is an important aspect of our lives. For businesses, knowing what is changing and how it has changed is also crucial. There are two main objectives for mining changes in a business environment:

1. To follow the trends: The key characteristic of this type of applications is the word "follow". Companies want to know where the trend is going and do not want to be left behind. They need to analyze customers' changing behaviors in order to provide products and services that suit the changing needs of the customers.
2. To stop or to delay undesirable changes: In this type of applications, the keyword is "stop". Companies want to know undesirable changes as early as possible and to design remedial measures to stop or to delay the pace of such changes. For example, in a shop, people used to buy tea and creamer together. Now they still buy tea, but seldom buy creamer. The shopkeeper needs to know this information so that he/she can find out the reason and design some measures to attract customers to buy creamer again.

In many applications, mining for changes can be more important than producing accurate models for prediction, which has been the focus of existing data mining research. A model, no matter how accurate, in itself is passive because it can only predict based on patterns mined in the old data. It should not lead to actions that may change the environment because otherwise the model will cease to be accurate.

Y. Kambayashi, M. Mohania, and A M. Tjoa (Eds.): DaWaK 2000, LNCS 1874, pp. 337–346, 2000.
© Springer-Verlag Berlin Heidelberg 2000

Building models for prediction is more suitable in domains where the environment is relatively stable and there is little human intervention (i.e., nature is allowed to take its course). However, in many business situations, constant human intervention to the environment is a fact of life. Companies simply cannot allow nature to take its course. They constantly need to perform actions in order to provide better services and products. For example, in a supermarket, there are always discounts and promotions to raise sale volume, to clear old stocks and to generate more sales traffic. Change mining is important in such situations because it allows the supermarket to compare results before and after promotions to see whether the promotions are effective, and to find interesting changes and stable patterns in customer behaviors.

Even in a relatively stable environment, changes (although in a slower pace) are also inevitable due to internal and external factors. Significant changes often require immediate attention and actions to modify the existing practices and/or to alter the domain environment. Let us see a real-life example.

A company hired a data mining consultant firm to build a classification model from their data. The model was built using the decision tree engine in a commercial data mining system. The accuracy was 81% at the time when the model was built. However, after the Asia financial crisis, the model only worked 60% of the time. The company asked the consultant why the classification model did not work any more. The reason, of course, is simple, i.e., the training data used to build the model (or classifier) was collected before the financial crisis. The consultant firm then built a new model (a classifier) for the company using the data collected after the financial crisis. The model was again accurate. However, after a while the company realized that the new model did not help. The reason is also simple. The company's profit is dropping and an accurate model could not stop this decline. What they really need is to know what has changed in the customer behaviors after the financial crisis so that they can perform some actions to reverse the situation. This requires change mining to compare the data collected from different periods of time.

In this paper, we study change mining in the contexts of decision tree classification. The study is motivated by two real-life data mining applications (see Section 3.2). In these applications, the users want to find changes in their databases collected over a number of years. In our literature search, we could not find suitable techniques to solve the problems. We thus designed a method for change mining in decision tree classification. This method has been incorporated into a decision tree algorithm to make it also suitable for mining changes.

There are existing works that have been done on learning and mining in a changing environment. Existing research in machine learning and computational learning theory has been focused on generating accurate predictors in a drifting environment [e.g., 14, 5, 7, 17]. It does not produce the explicit changes that have occurred. In data mining, [1, 4] addressed the problem of monitoring the support and confidence changes of association rules. [6] gave a theoretical framework for measuring changes. We will discuss these and the other related works in Section 4.

2. Mining Changes in the Decision Tree Model

Decision tree construction is one of the important model building techniques. Given a data set with a fixed discrete class attribute, the algorithm constructs a classifier of the

domain that can be used to predict the classes of new (or unseen) data.

Traditionally, misclassification error rate is used as the indicator to show that the new data no longer conforms to the old model. However, the error rate difference does not give the characteristic descriptions of changes, as we will see below. Additional techniques are needed.

In a decision tree, each path from the root node to a leaf node represents a hyper-rectangle region. A decision tree essentially partitions the data space into different class regions. Changes in the decision tree model thus mean changes in the partition and changes in the error rate (see Section 2.3). Our objective in change mining is:

- to discover the changes in the new data with respect to the old data and the old decision tree, and present the user with the exact changes that have occurred.

The discovered changes should also be easily understood by the user. Our application experiences show that changes are easily understood if they are closely related to the old decision tree or the old partition.

2.1 Approaches to change mining

Below, we present three basic approaches to change mining in the decision tree model: *new decision tree*, *same attribute and best cut*, and *same attribute and same cut*. The first two approaches modify the original tree structure, which makes the comparison with the original structure difficult. The third approach is more appropriate and it is the method that we use. Here, we discuss these three approaches. The detailed algorithm for the third approach is presented in Section 2.2. Note that the basic decision tree engine we use in our study is based on that in C4.5 [15]. We have modified it in various places for change mining purposes.

1. **New decision tree**: In this method, we generate a new decision tree using the new data, and then overlay the new decision tree on the old decision tree and compare the intersections of regions. The intersection regions that have conflicting class labels are the changes. This idea was suggested in [6].
2. **Same attribute and best cut**: This method modifies the decision tree algorithm so that in generating the new tree with the new data, it uses the same attribute as in the old tree at each step of partitioning, but it does not have to choose the same cut point for the attribute as in the old tree. If the algorithm has not reached the leaf node of a particular branch in the old tree and the data cases arrive here are already pure (with only one class), it stops, i.e., no further cut is needed. If any branch of the new tree needs to go beyond the depth of the corresponding branch in the old tree, the normal decision tree building process is performed after the depth.
3. **Same attribute and same cut:** In this method, we modify the decision tree engine so that in building the new tree, it not only uses the same attribute but also the same cut point in the old tree. If the algorithm has not reached the leaf node of a particular branch in the old tree and the data cases arrive here are already pure, it stops. If any branch of the new tree needs to go beyond the depth of the corresponding branch in the old tree, the normal process is performed.

In all three approaches, decision tree pruning is also performed. Let us use an example to show the differences among the three approaches. We use the iris data set from UC Irvine machine learning repository [12] in the example. The iris data has 4 attributes and 3 classes. We only use two attributes (and all 3 classes) here for illustration. The data points in the original data set are drawn in Figure 1 together with the partition produced by the decision tree engine. Next, we introduce changes in the

data by shifting some data points of setosa class in region 1 (Figure 1) toward left and put some versicolor class points in the space vacated, see the shaded area in Figure 2.

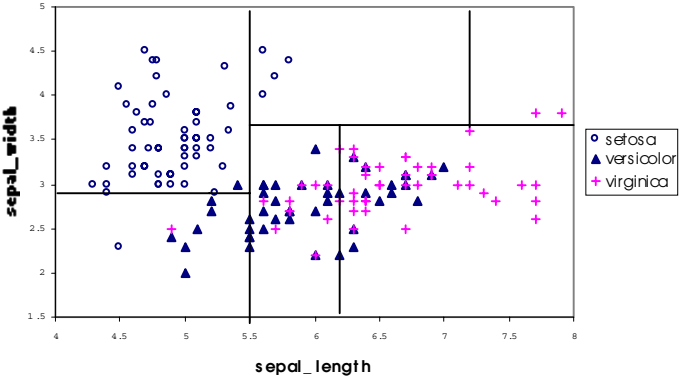


Fig. 1. Partition produced by decision tree on the original data

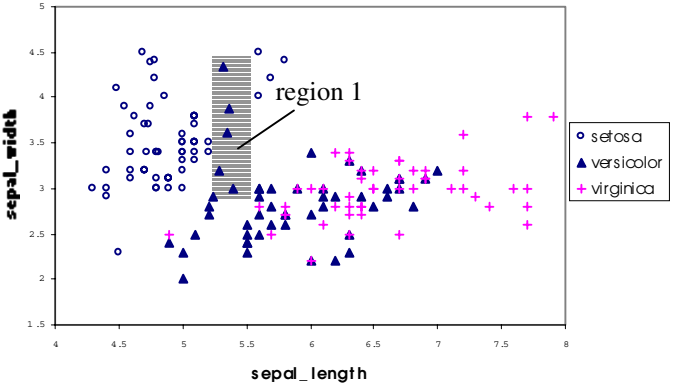


Fig. 2. The introduced change

We look at approach 1 first. Figure 3 shows the partition produced by approach 1 on the new data after the changes have been introduced. From Figure 3 alone, it is not clear what the changes are.

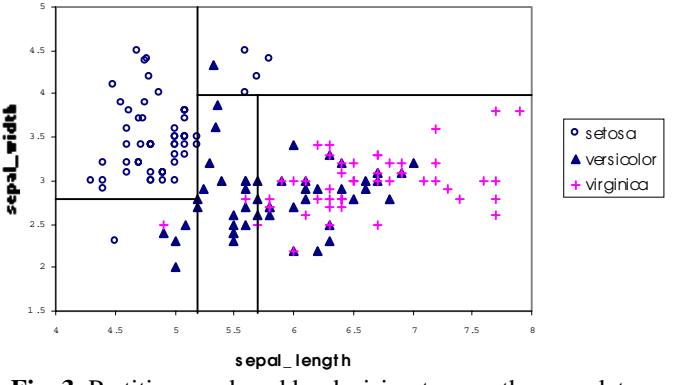


Fig. 3. Partition produced by decision tree on the new data

Figure 4 shows the overlay of this partition (Figure 3) on the old partition (Figure 1). Dashed lines represent the old partition (produced from the old data). The shaded areas are the conflicting regions, which represent changes.

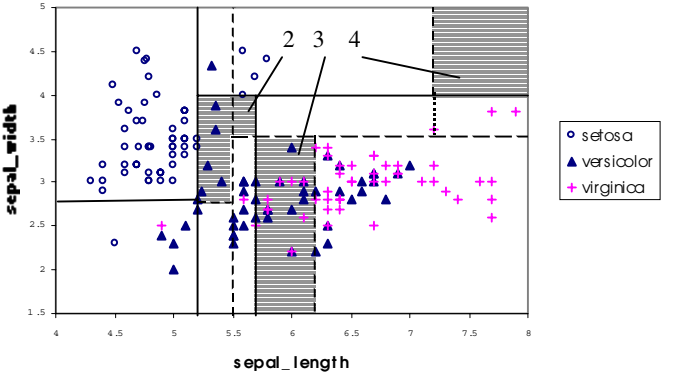


Fig. 4. Overlay of the new partition on the old partition

Clearly, the result is not satisfactory. It produces changes that do not actually exist, i.e., the intersection areas marked 2, 3, and 4. These problems become more acute when the number of attributes involved is large. The reason is that the changes in data points caused the decision tree engine to produce a completely new partition that has nothing to do with the old partition (it is well known that the decision tree algorithm can result in a very different tree even if only a few data points are moved slightly).

The same problem exists for the second approach, same attribute and best cut. Figure 5 shows the new partition. Although the algorithm tries to follow the attribute sequence in the old tree, but the cut points can change drastically, which make it hard to compare with the old tree. Again we need to use overlay to find changes. Hence, it has the same problem as the first approach.

The third approach (same attribute and same cut), on the other hand, does not have the problems. Figure 6 shows the partition obtained. The shaded area represents the change, which is precisely what has been introduced (see Figure 2). The change is also closely related to the old tree, and thus easily understood. We present the details of the third approach below, which is what we use in our system.

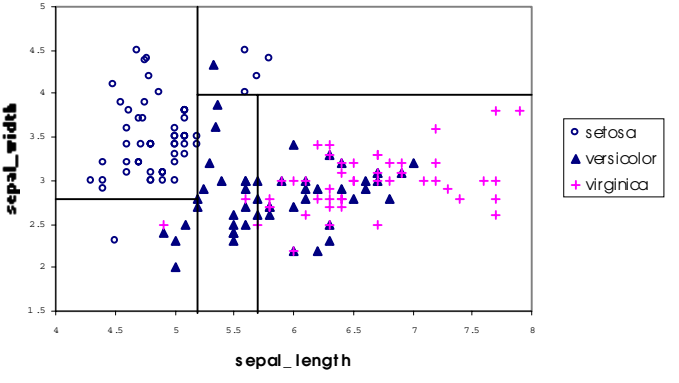


Fig. 5. Partition produced by the second approach

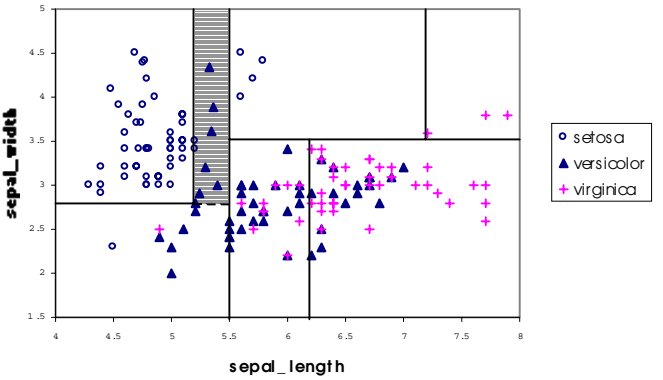


Fig. 6. Partition produced by the third approach

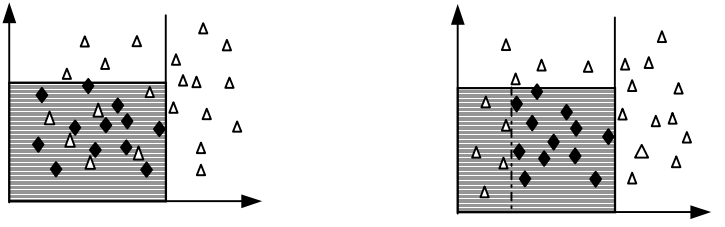
2.2. Change mining for the third approach: same attribute and same cut

The conceptual algorithm for the third approach is given in Figure 7. Let O_T be the old tree, and N_D be the new data. Let the decision tree building algorithm be buildTree().

Algorithm mineChange(O_T, N_D)
1 Force buildTree(N_D) to follow the old tree O_T (both the attributes and cut points) and stop earlier if possible;
2 Test for significance of the error rate change at each old leaf node and show the user those leaves that have significant error rate changes;
3 Grow the new tree further (not only those branches that have changed significantly);
4 Prune the tree to remove those non-predictive branches;
5 Traverse the new tree and compare it with O_T to identify changes;

Fig. 7. The proposed algorithm

Five points to note about the algorithm:
a. In line 1, the algorithm basically tries to follow the old tree.
b. Line 2 uses chi-square test [13] to test the significance of the changes in error rates.
c. In line 3, we allow the decision tree algorithm to grow the new tree further, not only those branches that have significant changes. The reason is that those branches that do not result in significant changes may be further partitioned to result in more homogenous regions, which is not possible in the old data.



(a) Partition from the old data (b) Partition from the new data

Fig. 8. Partitioning the region further

For example, in the old data in Figure 8(a), the shaded region cannot be

partitioned further because the error points (Δ) are randomly positioned. However, in the new data, the error rate for the shaded region remains the same as in the old partition, but it can now be further refined into two pure regions (Figure 8(b)).

d. In line 4, the new tree is subjected to pruning. Pruning is necessary because the new tree can grow much further and result in overfiting the new data. As in the normal decision tree building, we need to perform pruning to remove those non-predictive branches and sub-trees. There are two ways to perform pruning in normal decision tree building [15]. They are:
   1) discarding one or more sub-trees and replacing them with leaves;
   2) replacing a sub-tree by one of its branches.

   For change mining, the pruning process needs to be restricted. We still use (1) as it is because it only results in join of certain regions, which is understandable. However, (2) needs to be modified. It should only prune till the leaf nodes of the old tree. Those nodes above will not be subjected to the second type of pruning because otherwise it can result in drastic modification in the structure of the old tree, thus making the changes hard to identify and to understand.
e. In line 5, the algorithm traverses the new tree and compares it with the corresponding nodes in the old tree to report the changes to the user. Below, we present the types of changes that we aim to find using the proposed approach.

2.3 Identify different types of changes

There are many kinds of changes that can occur in the new data with respect to the old data. Below, we identify three main categories of changes in the context of decision tree building: *partition change*, *error rate change*, and *coverage change*. The first four types of changes presented below are partition changes, type 5 is the error rate change and type 6 is the coverage change. Their meanings will be made clear later.

Type 1. **Join of regions**: This indicates that some cuts in the old tree (or partition) are no longer necessary because the data points in the new data set arrive in the split regions are now homogeneous (of the same class) and need no further partitioning.

Type 2. **Boundary shift**: This indicates that a cut in the old tree is shifted to a new position. It only applies to numeric attributes. Boundary shifts only happen at those nodes right above the leaf nodes of the old tree. It is not desirable to allow boundary shifts in the earlier nodes because otherwise, the whole tree can be drastically changed resulting in the problem discussed in Section 2.1.

Type 3. **Further refinement**: This indicates that a leaf node in the old tree can no longer describe the new data cases arrive at the node (or the region represented by the node). Further cuts are needed to refine the node.

Type 4. **Change of class label**: This indicates that the original class of a leaf node in the old tree has changed to a new class in the new tree. For example, a group of people who used to buy product-1 now tend to buy product-2.

Partition changes basically isolate the changes to regions (which can be expressed as rules). They provide the detailed characteristic descriptions of the changes. They are very useful for targeted actions. Though useful, partition changes may not be sufficient. Error rate change and coverage change are also necessary. The reasons are two-fold. First, sometime we cannot produce a partition change because the changes of the data points are quite random, i.e., we cannot isolate the changes to some regions. Nevertheless, there is change, e.g., the error rate has increased or decreased,

or the proportion of data points arrives at a node has increased or decreased. Second, even if we can characterize the changes with regions, the data points in the regions may not be pure (i.e., with different classes of points). In such cases, we need error rate change to provide further details to gauge the degree of change.

Type 5. Error rate change (and/or class distribution change): This indicates that the error rate of (and/or the class distribution of the data points arrive at) a node in the new tree is significantly different from the error rate of the same node in the old tree. For example, the error rate of a node in the old tree is 10%, but in the corresponding node of the new tree it is 40%.

Type 6. **Coverage change**: This indicates that the proportion of data points arrives at a node has increased or decreased significantly. For example, in the old tree, a node covers 10% of the old data points, but now it only covers 2% of the new data points.

The partition changes (the first four types) can be easily found by traversing the new tree and comparing it with the old tree. The information on the last two types of changes is also easily obtainable from the final new tree and the old tree.

3. Experiments and Applications

We evaluate the proposed techniques using synthetic data sets and real-life data. The goal is to assess the effectiveness of the proposed technique (the third approach in Section 2.1). Efficiency is not an issue here because it uses an existing technique for decision tree building [15], which has been proven very efficient.

3.1 Synthetic data test

We implemented a synthetic data generator to produce data sets of 2 classes. It takes as input, the number of attributes (all attributes are numeric attributes), range of values for each attribute, the number of data regions with attached classes, the locations of the data regions to be generated, and the number of data points in each region. Basically, the data generator generates a number of regions with data points in them, and each data point is also labeled with a class. Each region is in the shape of a hyper-rectangle (each surface of the hyper-rectangle is parallel to one axis and orthogonal to all the others). The data points in each region are randomly generated using a uniform distribution. In the new data generation, we introduce changes by modifying the input parameters used for the old data set generation.

In our experiments (the results are summarized in Table 1), we used data sets with 3, 5, 8 and 10 dimensions. We first generate 3 old data sets (i.e., No. of Expts in Table 1) with 6, 8 and 10 data regions for each of the 3, 5, 8 and 10 dimensional spaces. For each old data set, we then generate one new data set and introduce all 6 types of changes at different locations. We then run the system to see if the planted changes can be identified. Experiment results show that all the changes embedded are found.

Table 1. Synthetic data test

No. of Expts	No. of dimensions	Types of changes introduced	Changes found
3	3	6	✔
3	5	6	✔
3	8	6	✔
3	10	6	✔

3.2 Real-life Data Tests

We now describe our two real-life data tests. Due to confidentiality agreement, we could not disclose the detailed findings. The first application is for an educational institution that wishes to find out more about its students. We were given the data from the past few years. The data includes the student's examination results, family background, personal particulars etc. Our user was interested in knowing how the performances of different groups of students had changed over the years. This requires change mining. A base year was chosen to build the old tree and the program was run by comparing the subsequent years with the base year decision tree. Some interesting trends immediately became apparent using our technique. For instance, we were able to tell our user that the performances of certain group (with some characteristics) of students in a particular subject had steadily deteriorated over the years. In addition, we also discovered that in a particular year, a group of students suddenly outperformed another group that had consistently been the better students.

The second application involves the data from an insurance company. The user knew that the number of claims and the amounts per claim had risen significantly over the years. Yet, it was not clear whether there were some specific groups of people who were responsible for the higher number of claims or that the claims were just random. In order to decide suitable actions to be taken, our user wanted to know what are the claiming patterns of the insurers over the years. Using data from the past five years, our system again discovered some interesting changes. We found that certain groups of insurers had gradually emerged to be the major claimers over the years. On the other hand, there exists another group of insurers that no longer had any claims even though in the beginning, they did put in claims.

4. Related Work

Mining and learning in a changing environment has been studied in machine learning [e.g., 14, 17], data mining [1, 4] as well as in computational learning theory [5, 7]. In machine learning, the focus is on how to produce good classifiers in on-line learning of a drifting environment [14, 17]. The basic framework is as follows: The learner only trusts the latest data cases. This set of data cases is referred to as the *window*. New data cases are added to the window as they arrive, and the old data cases are deleted from it. Both the addition and deletion of data cases trigger modification to the current concepts or model to keep it consistent with the examples in the window. Clearly, this framework is different from our work as it does not mine changes.

In computational learning theory, there are also a number of works [5, 7] on learning from a changing environment. The focus is on theoretical study of learning a function in a gradually changing domain. They are similar in nature to those works in machine learning, and they do not mine changes as the proposed method does.

[6] presented a general framework for measuring changes in two models. Essentially, the difference between two models (e.g., two decision trees, one generated from data set D_1 and one generated from data set D_2) is quantified as the amount of work required to transform one model into the other. For decision trees, it computes the deviation by overlaying of two trees generated from two data sets respectively. We have shown that overlaying of one tree on another is not satisfactory for change mining. The proposed method is more suitable.

Another related research is subjective interestingness in data mining. [10, 11, 16] gives a number of techniques for finding unexpected rules with respect to the user's existing knowledge. Although the old model in our change mining can be seen as "the user's existing knowledge" (it is not from the user), interestingness evaluation techniques cannot be used for mining changes as its analysis only compares each newly generated rule with each existing rule to find the degree of difference. It does not find which aspects have changed and what kinds of changes have occurred.

5. Conclusion

In this paper, we study the problem of change mining in the contexts of decision tree classification. This is motivated by two real-life applications. We could not find an existing technique to solve the problems. This paper proposed a technique for the purpose. Empirical evaluation shows that the methods are very effective.

We believe change mining will become more and more important as more and more data mining applications are implemented in production mode. In our future work, we plan to address the problem of evaluating the quality of the changes and to detect reliable changes as early as possible by on-line monitoring.

References

[1] Agrawal, R. and Psaila, G. "Active data mining." *KDD-95*, 1995.
[2] Agrawal, R., Imielinski, T., Swami, A. “Mining association rules between sets of items in large databases.” *SIGMOD-1993*, 1993, pp. 207-216.
[3] Cheung, D. W., Han, J, V. Ng, and Wong, C.Y. “Maintenance of discovered association rules in large databases: an incremental updating technique.” *ICDE-96*, 1996.
[4] Dong, G. and Li, J. “Efficient mining of emerging patterns: discovering trends and differences.” KDD-99, 1999.
[5] Freund, Y and Mansour, Y. “Learning under persistent drift” Computational learning theory: Third European conference, 1997.
[6] Ganti, V., Gehrke, J., and Ramakrishnan, R. "A framework for measuring changes in data characteristics" *POPS-99*.
[7] Helmbold, D. P. and Long, P. M. “Tracking drifting concepts by minimizing disagreements.” *Machine Learning*, 14:27, 1994.
[8] Johnson T. and Dasu, T. "Comparing massive high-dimensional data sets," *KDD-98*.
[9] Lane, T. and Brodley, C. "Approaches to online learning and concept drift for user identification in computer security." *KDD-98*, 1998.
[10] Liu, B., Hsu, W., “Post analysis of learnt rules." *AAAI-96*.
[11] Liu, B., Hsu, W., and Chen, S. “Using general impressions to analyze discovered classification rules.” *KDD-97*, 1997, pp. 31-36.
[12] Merz, C. J, and Murphy, P. UCI repository of machine learning databases [http://www.cs.uci.edu/~mlearn/MLRepository.html], 1996.
[13] Moore, D.S. “Tests for chi-squared type.” In: R. B. D’Agostino and M. A. Stephens (eds), *Googness-of-Fit Techniques*, Marcel Dekker, New York, 1996, pp. 63-95.
[14] Nakhaeizadeh, G., Taylor, C. and Lanquillon, C. “Evaluating usefulness of dynamic classification”, *KDD-98*, 1998.
[15] Quinlan, R. *C4.5*: *program for machine learning.* Morgan Kaufmann, 1992.
[16] Silberschatz, A., and Tuzhilin, A. “What makes patterns interesting in knowledge discovery systems.” *IEEE Trans. on Know. and Data Eng.* 8(6), 1996, pp. 970-974.
[17] Widmer, G. "Learning in the presence of concept drift and hidden contexts." *Machine learning*, 23 69-101, 1996.

AIM: Approximate Intelligent Matching for Time Series Data

Edward D. Kim, Joyce M.W. Lam, Jiawei Han

School of Computing Science, Simon Fraser University
British Columbia, Canada, V5A 1S6
{kime, lamd, han}@cs.sfu.ca

Abstract. Time-series data mining presents many challenges due to the intrinsic large scale and high dimensionality of the data sets. Subsequence similarity matching has been an active research area driven by the need to analyse large data sets in the financial, biomedical and scientific databases. In this paper, we investigate an intelligent subsequence similarity matching of time series queries based on efficient graph traversal. We introduce a new problem, the *approximate partial matching* of a query sequence in a time series database. Our system can address such queries with high specificity and minimal time and space overhead. The performance bottleneck of the current methods were analysed and we show our method can improve the performance of the time series queries significantly. It is general and flexible enough to find the *best approximate match* query without specifying a tolerance ε parameter.

1 Introduction

Subsequence similarity matching in time series data suffers from the *curse of dimensionality*, since a query sequence may be matched to any offset in the sequence database. A popular method to address this problem, is to use an orthogonal transform (such as FFT[1], wavelets[4]) to map the data points to a feature space, and extracting features to index the sequence into a spatial index. There has been significant efforts in finding similar sequences using linear transformations [9, 12], probabilistic methods [10], and episodic events [11].

This paper focuses on *approximate similarity* matching of a user-defined query subsequence in a sequence database. We introduce the notion of *partial* similarity matching of the query, in which we attempt to match significant features of the input query sequence. Obviously, a partial match of a short query section is relatively unimportant. In the extreme case, a single point may match but this is not significant and is likely of little importance to the user. A maximal partial match of the query may be the most significant to user, since this identifies the major features of the query. Our method optimizes the expensive post-processing stage of the current *search-and-prune* methods [7, 4, 12], so it can improve the performance of all these methods.

The outline of the paper is as follows: we begin with a background of the time series similarity problems and the current methods. An analysis of the ST-index

Y. Kambayashi, M. Mohania, and A M. Tjoa (Eds.): DaWaK 2000, LNCS 1874, pp. 347–357, 2000.
© Springer-Verlag Berlin Heidelberg 2000

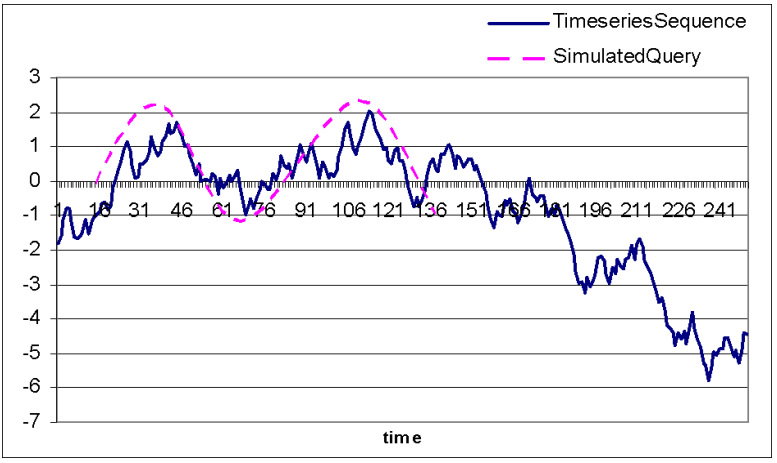


Fig. 1. Random walk sequence and Simulated user-defined query

method [7] leads us to the motivation of our study. We present some interesting problems that are not feasible with the current technology, and we propose our method of addressing the approximate matching, based on efficient graph traversal. We demonstrate the effectiveness of our system with performance experiments, followed by a discussion of the results.

2 Background

An early work in time-series *subsequence similarity* matching introduced the ST-index [7] which combined feature mapping with spatial indexing (further details given below). Agrawal et. al. [2] defined subsequence similarity as non-overlapping piecewise matching segments with minimal gaps. They used a window stitching algorithm to ignore small gaps where there may be an outlier. A partial match can be considered a long segment match with an infinite length gap, but it is not clear how the above methods can be adapted to efficiently handle this.

The distinction between *exact* and *approximate* similarity matching was formalized for time-series data in [9], and the definition of similarity was extended to include shape-preserving transformations like normalization, shifting and scaling. Subsequently, other transformation-based similarity queries such as inverting, moving average [12], and warping [13] have been presented. Since those techniques are typically performed before the spatial index search, and we address the post-processing stage, our method is complementary to these transforms and may be used in conjunction with these methods.

In current similarity search methods, there is a heavy reliance on the user-specified tolerance ε. The query results and performance are intrinsically tied to this subjective parameter, which is a real usability issue. We give an *approximate best match* query that eliminates this requirement from the user. This is similar

Table 1. Summary of symbols and their definitions.

Symbols	Definitions
S	A time-series sequence.
Q	A query sequence.
w	FFT window size; same as $\|FFT\|$.
N	Number of subqueries (or query segments) in Q.
w	the subquery and subtrail window size.
$S[i]$	A point i in the sequence S.
$Q[i]$	A point i in the query Q.
$D(S,Q)$	Euclidean distance between S and Q of equal lengths;
s_i	The i-th subsequence in a time-series sequence.
q_i	The i-th subquery (or query segment) in a query sequence.
$s_i[j]$	The j-th point in s_i subsequence.
$q_i[j]$	The j-th point in q_i subquery.
C_j	A candidate subsequence containing candidate segments which match a particular subquery.
s_j^{qi}	A candidate segment in C_j for subquery q_i.
n	Number of query segments matching a subsequence in partial matching.
ε	Error tolerance for similarity matching.
ε_u	Unit error tolerance; the maximum error allowed when matching s_i against q_i
ε_α	Proportional error tolerance; the maximum error allowed in a partial match.

to the nearest neighbour query mentioned in [4], but their algorithm considers only single piece queries. Handling long multi-piece queries face the same pruning inefficiencies as the ST-index.

Analysis of ST-index We chose to analyse the ST-index since its *search-and-prune style* is the basis for many of the current methods in similarity search [9, 4, 12, 13]. A basic sketch of the ST-index is provided here but we encourage the reader to review the details in [7]. First, the input sequence S is mapped from the time domain to frequency domain using the fast fourier transform (FFT). By taking the first few FFT coefficients (2-3) of a sliding window w over the data sequence, the time series is transformed into a feature trail in multidimensional space (figure 2). The trails are divided into sub-trails which are subsequently represented by a Minimum Bounding Rectangle (MBR) and stored in a spatial index structure, the R*tree[3].

A sequence query Q is decomposed into query segments $q_1, q_2, \cdots q_N$ of window size w. Each query segment q_i is mapped to a point in feature space, and used as a multidimensional point query into the R*tree (figure 3). A point subquery search performed on the R*tree returns all the intersecting MBR's. This results in a set of possible candidates for each sub-query.

For instance, in figure 4, a query Q with N subquery segments of window size $w = 100$, gives a list of sub-trails in the sequence database identified by the

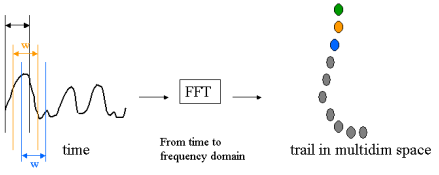


Fig. 2. Feature mapping in ST-index

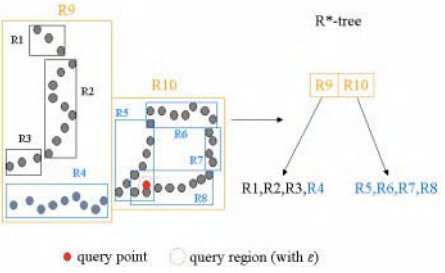


Fig. 3. Point query search in ST-index

starting point offset in the sequence; that is, the actual matching segment begins somewhere in that sub-trail.

In the ST-index, the Euclidean distance $D(S,Q) < \varepsilon$ is used to compare the subsequences against the query to discard the false alarms.

$$D^2(S,Q) = \sum_{i=1}^{n} (S[i] - Q[i])^2 < \varepsilon^2 \tag{1}$$

For a query with error tolerance ε, a matching subsequence must have at least one subtrail s_i within $\varepsilon/\sqrt{N}$ radius of the multidimensional sub-query q_i point; the Lemma is given in [7].

$$D^2(s_i, q_i) = \sum_{j=i*w+1}^{(i+1)*w} (s_i[j] - q_i[j])^2 < \varepsilon^2/N \tag{2}$$

We call this the *unit error tolerance* ε_u. The subtrail found to be within ε_u of the subquery is a candidate for a full query match. Depending on how tight the ε bound is specified, the subquery check reduces the candidate set and then a post-processing check of the full query against the raw time series data is performed, to finally remove all the false positives and find the true subsequences which match the query.

3 Motivation

The ST-index performance can be broadly broken down into two steps: the spatial index (R*tree) search, and the post-processing stage. Spatial and high-dimensional indexing is a very active research area [8] so any advancements there can be applied to the spatial indexing stage. Yet, there has not been much attention given to the post-processing phase of the similarity search. We believe the current methods are not particularly optimized for longer approximate query matching. In fact, we hypothesize that under realistic conditions the post-processing time may be the bottleneck in the ST-index. Specifically, the post-processing step where each candidate from the R*tree search must be checked as a potential match can be a very time consuming process.

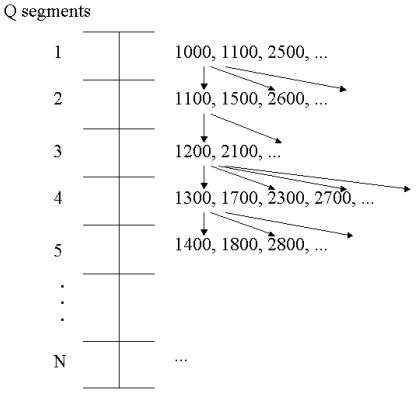


Fig. 4. Candidate sets returned from subquery searches

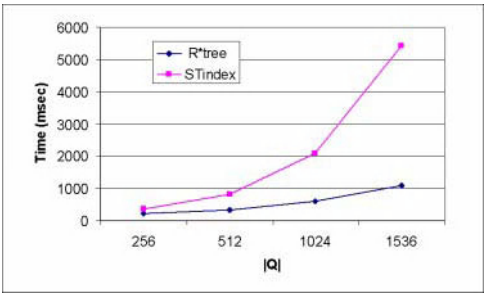


Fig. 5. ST-index, R*tree times vs. $\Delta|Q|$, $|S| = 20000$, $|FFT| = 32$

We ran performance studies to identify the spatial index and post-processing times required for the similarity matching. For a random walk sequence S of length 20000 points with mean= 0 and variance= 1, the tolerance was set to $\varepsilon = \sqrt{N}$ to allow sufficient candidate matches. Notice, in figure 5, as the query length Q gets longer, the post-processing phase contributes to the majority of the overall ST-index time. Further, the post-processing time after the R*tree search becomes proportionally higher. This can be explained by the fact that a longer query has more segment pieces q_i, and since each subquery search in the R*tree returns a set of candidate segments, there are more candidates on which false positive checks must be made. Hence, for a long query, the post-processing time to remove the false positives dominates the total time for the ST-index. Similarly, queries with larger tolerance ε generated more candidates for post-processing and scaled poorly.

We were motivated to study approximate similarity matching of long queries by the inflexibility of the current query methods. A long time-series query with multiple features (like double peaks shown in Figure 1) may not have a match within a given tolerance, in the time-series database. The user may have specified the error tolerance too restrictively, which does not allow a close but inexact match to be uncovered.

Partial Query Match Given a user-supplied query Q and error $\varepsilon > 0$, we seek a *partial approximate match* in the time-series database. Although there may not be a match for the full query Q in the sequence database, we would like to identify the subsequences which match a *significant* portion of the query.

Suppose a subsequence has a significant portion of the query matching but the non-matching segment causes the distance metric to surpass the allowable tolerance. It may be preferable to return to the user some answers that are partial matches of the input query rather than not returning any response at all. This

is analogous to intelligent query answering in relational databases, and we feel it has relevance with time-series data.

As shown above, the query performance and results can be substantially affected by the user's choice of ε. Even though the user may not be aware of how the query length is related to the error tolerance, s/he is required to input such a subjective parameter. We feel this is putting too much of a burden on the user who wants to perform a subsequence query, but has no intimate knowledge of the data scale or distance metric.

Best Approximate Match Given an input query Q and without an error tolerance ε, the Best Approximate Matches are the subsequences in the sequence database S which has the N lowest Euclidean errors from the input query Q.

This would be a more realistic scenario, where the user just enters a query and requests the top N matches in the sequence database. Now, all the candidates returned from the R*tree search are potentially a "best match".

4 Proposed Method

We observed the fact that the set of candidates returned from the spatial index (R*tree) search form a directed acyclic graph (DAG). Thus, if we consider the subtrails as nodes in a graph, a matching subsequence corresponds to the longest path in the DAG. Hence, we can apply an efficient graph traversal method to the candidate sets to determine the partial or best match. Now, the similarity match can be *reformulated as a longest path in a DAG* problem with an error cost function. Depth-first search (DFS) is known to be an optimal solution to the longest path problem in a DAG [5].

Also we exploit the DFS node-coloring technique whereby if a node has been visited, then the node is colored so there is no need to traverse down that node again. This quickly reduces the branching factor of the candidate sets, which leaves very few candidates for the expensive distance measurement against the query sequence.

There are two kinds of windows in this algorithm; the FFT window which is a sliding window that generates a trail in feature space, and the subtrail windows (represented by s_j^{qi} in the algorithm below) which are adjacent, sequential windows. Without loss of generality, let the FFT window size equal the subtrail window size, $|FFT| = |s_j^{qi}|$. Our algorithm will also work if the subtrail window size is some multiple of the FFT window size. There will be a higher branching factor from each node but in general our algorithm is still applicable. For demonstrative purposes, we will assume $|FFT| = |s_j^{qi}|$ throughout rest of the paper. The algorithm is given next and we explain why our method works.

DFS Algorithm

1. A query Q is decomposed into query segments $q_1, q_2, \cdots q_N$, and each query segment q_i is used as a multidimensional point query into the R*tree.

2. Then each subquery q_i returns a set of candidate subsequences C_i from the R*tree:

$$\begin{bmatrix} q_1 \\ q_2 \\ \vdots \\ q_N \end{bmatrix} \Rightarrow \begin{bmatrix} C_1 \\ C_2 \\ \vdots \\ C_N \end{bmatrix} = \begin{bmatrix} \{s_1^{q1}, s_2^{q1}, \cdots, s_x^{q1}\} \\ \{s_1^{q2}, s_2^{q2}, \cdots, s_y^{q2}\} \\ \vdots \\ \{s_1^{qN}, s_2^{qN}, \cdots, s_z^{qN}\} \end{bmatrix}$$

   The superscript refers to the subquery q_i that generated this candidate set. The subtrail segments form nodes in the candidate graph shown in figure 4.
3. For each candidate segment s_j^{qi}, we find the lowest point of error compared to the subquery piece q_i. Since the subquery piece and the segment window is small, a distance check to find the lowest error in that segment is relatively fast. We check that the subtrail is within unit error ε_u.
4. To find the longest matching subsequence, we use a greedy approach to expand the matching segment. In other words, we expand the subsequence in the direction with lowest segment error. We keep track of the total error to see that accumulative error for the subsequence is within the proportional error ε_α (see below). The matching subsequence is expanded out in the direction of lowest error until sum of all ε_u is greater than ε_α or the query size $|Q|$ is reached. We store the matching offset information in an array sorted by the distance error.
5. For another candidate segment s_b^{qa}, if the starting point of this segment is already included a previous subsequence match, then there is no need to match this subtrail. We can skip the longest path search for this segment, which has the same effect of skipping a visited node on a DFS graph traversal. This results in very fast pruning of candidate nodes in the longest path search.

Typically a graph search is considered an exhaustive operation, but in our scenario, DFS is very efficient and much faster than going back to the time-series and performing a post-processing sequential match on the raw data. Although there are many nodes (ie. candidate segments) in our graph, there is, in fact, a low branching factor from any one node to the next.

Proportional Error Tolerance The above algorithm uses the *unit error* ε_u and the *proportional error* ε_α which we define here. If we uniformly distribute the allowable error ε for the query, the unit error for a query divided into N segments is $\varepsilon_u = \varepsilon/\sqrt{N}$. By extension, the proportional error for n subquery segments is

$$\begin{aligned} \varepsilon_\alpha &= (n/N \times \sum_{i=1}^{nw} (S[i+K] - Q[i])^2)^{1/2} \\ &= \varepsilon \times \sqrt{n/N} \end{aligned}$$

where K is the offset into the matching sequence. Now we can match a portion of the query length within a proportional error tolerance. To clarify, the proportional tolerance is not a static measure but is scaled for the number of subquery segment under consideration.

A partial match then is defined as a subsequence that matches a subsection of the query within a *proportional error tolerance* ε_α. The proportional error tolerance is the allowable error for a partial match. For example: if the longest partial match is half the length of the query, the allowable error is $\varepsilon_\alpha = \varepsilon/\sqrt{2}$.

For the partial match case where ε is given, the DFS is started at a segment where a match is within ε_u. With the ST-index we have all the segment errors in a vector, so it is possible to keep track of whether the cumulative error is greater than the proportional error ε_α. For the best match query, start at the segment with the lowest unit error, and take the most consecutive candidate segments and test for errors.

5 Experiments

We report two experiments to demonstrate the effectiveness of our method and compare with existing methods: a partial match query and a best approximate match query. Our method is compared against the ST-index method for each of these experiments. We do not show the naive sequential search times since they have much longer run times [7] than any of these methods and may distort the comparative graphs. Since our DFS method and the ST-index both implement a spatial index (R*tree) search initially, we show the R*tree search time to illustrate the post-processing cost. For all our experiments, we took the average of 10 runs to smooth the random variations. Our system and ST-index prototype was written in Java 1.2.2 on Windows NT 4.0 running on an Intel Pentium 166MHz processor with 96 MB of RAM.

Partial Match Experiment For the partial match, we keep extending the longest path in the graph while the proportional error ε_α is within the allowable error tolerance. Even though the longest path may not be the full length of the query, we can find the longest partial match which the user may be interested to discover. In the partial match experiment, our hypothesis is that we expect to find all the correct matches (100% specificity) with no major overhead. The experiment is designed as follows:

- Two random subsequences are chosen from the sequence database, and we call these query pieces Q_1 and Q_2.
- They are smoothed using a 16-point moving average, since we wish to simulate a smooth user-defined shape. This will also introduce some error such that it will result in more than one possible match.
- For each query, calculate the error from the correct positions, and make the ε error tolerance larger to allow other possible matches.
- Then concatenate the two query subsequences Q_1 and Q_2 into one query Q.

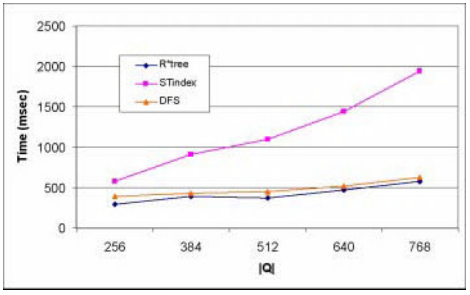


Fig. 6. Scalability of Partial Match vs. $\Delta|Q|$, $|S| = 20000, |FFT| = 32$

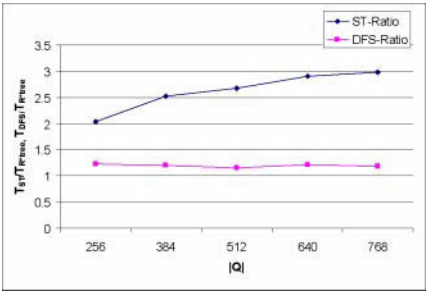


Fig. 7. Scalability of Best Approximate Match vs. $\Delta|Q|$, $|S| = 20000, |FFT| = 32$

– Partial match query is run to find both portions of the query. That is, we expect to find both Q_1 and Q_2 as the top two matches using the DFS algorithm.

We sought a comparison for our partial approximate match query, but we were not aware of any existing methods which were capable of handling such queries. We decided to compare these results to the times of the ST-index search, even though a correct match is not expected. An ST-index query with the composite query Q can have variable results. It may find one of the two sub-queries (ie. Q_1 or Q_2), or find another subsequence which does not contain either Q_1 or Q_2 but somehow satisfies the ε tolerance and is considered a match. However, these results are not reliable nor predictable and were not considered significant for our purposes. We will just compare the running times for an ST-index query against the DFS partial match query.

Notice in figure 7, our method only contributes to a small incremental time beyond the R*tree search time and outperforms the ST-index time as the query size $|Q|$ increases. Our DFS method for post-processing was able to identify the subqueries with 100% specificity, and in comparison with the ST-index, there is very little overhead above the R*tree search time. Hence, we can find partial match queries with minimal overhead and high reliability.

Best Approximate Match Experiment The ST-index can be modified to find the best match by finding the lowest matching error among the candidate sets. Since there is no ε defined, the subtrail pruning step would be very inefficient, as all candidates would have to be considered. There would be a sorting step after the post-processing to find the subsequence with the lowest error.

– For a best approximate match experiment without ε, create a random walk query, and smooth it using 16-point moving average. A random walk was chosen to ensure there would be more than one clear best match.
– Find the top N best matches using the error measurement for DFS and ST-index. Compare the result sets from both methods.

The DFS algorithm used here is essentially the same as in the partial match experiment, but there is no ε to stop the expansion of the subsequence pieces. Hence, the comparative times will be similar to that in figure 6. For this experiment, we give the run times as a ratio of the ST-index time. Again, since both methods use the R*tree search, the ratio T_{ST}/T_{R*tree} gives the overhead cost of the ST-index post-processing. In figure 7, the DFS method shows almost a fixed ratio overhead, while the ratio of the ST-index times is growing with the query length. It is worthwhile to remember, the spatial index times also grows with query length, so when the ratio grows, the run times become prohibitively expensive.

6 Concluding Remarks

We were motivated to study approximate time-series matching since current subsequence matching algorithms, like ST-index, do not consider flexible matching scenarios. In this light, we present two realistic problems that have not yet been addressed adequately, the partial match and best approximate match problems. Our experiments show an effective solution clearly addressing the post-processing inefficiencies of the existing methods.

As mentioned earlier, time-series similarity search has been an active area of research. In spite of these many efforts, there are other interesting similarity search problems that have not been addressed. In this study, we extended the notion of approximate similarity matching defined in [9]. We gave efficient solutions to the partial and best approximate match queries. Our method can also be used to address the partial best approximate match (without ε). In this scenario, the longest path with the lowest average unit error is considered the best partial match. But there is some issue of how to compare partial matches of different lengths. A subsequence with a longer partial match and a higher average unit error may be preferable to a shorter segment with a low unit error. The preference would be essentially application-dependent and may be addressed with a heuristic function of the partial match length and the average unit error.

The major contribution of our work is allowing flexible query answering, while improving the efficiency of the time-series similarity query processing. Our DFS method provides effective solutions for the partial match and best approximate match queries. We call these intelligent matching schemes since it unburdens the user of having to fully specify a time series query. In the future, we seek to extend the intelligence from pattern matching to time-series pattern discovery. There has been some initial work in the direction of rule discovery [6], yet we feel this is a ripe field for further advancement in time-series data mining.

Acknowledgements The authors would like to thank Christos Faloutsos and M. Ranganathan for the use of their ST-Index implementation, and David Whitlock for his Java implementation of the R*-tree.

References

1. R. Agrawal, C. Faloutsos, and A. Swami. Efficient similarity search in sequence databases. In *Proc. 4th Int. Conf. Foundations of Data Organization and Algorithms*, Chicago, Oct. 1993.
2. R. Agrawal, K.-I. Lin, H. Sawhney, and K. Shim. Fast similarity search in the presence of noise, scaling, and translation in time-series databases. In *Proc. 1995 Int. Conf. Very Large Data Bases (VLDB'95)*, pages 490–501, Zurich, Switzerland, Sept. 1995.
3. N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In *Proc. 1990 ACM SIGMOD Int. Conf. Management of Data (SIGMOD'90)*, pages 322–331, Atlantic City, NJ, June 1990.
4. K. Chan and A. Fu. Efficient time series matching by wavelets. In *Proc. 1999 Int. Conf. on Data Engineering (ICDE'99)*, pages 126–133, Sydney, Australia, Mar. 1999.
5. T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. The MIT Press, Cambridge, Massachusetts, 1990.
6. G. Das, K. Lin, H. Mannila, G. Renganathan, and P. Smyth. Rule discovery from time series. In *Proc. 1998 Int. Conf. Knowledge Discovery and Data Mining (KDD'98)*, pages 16–22, New York, NY, Aug. 1998.
7. C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. In *Proc. 1994 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'94)*, pages 419–429, Minneapolis, Minnesota, May 1994.
8. V. Gaede and O. Günther. Multdimensional access methods. *ACM Comput. Surv.*, 30:170–231, 1998.
9. D. Q. Goldin and P. C. Kanellakis. On similarity queries for time-series data: Constraint specification and implementation . In *Proc. 1995 Int. Conf. Principles and Practice of Constraint Programming*, pages 137–153, Marseilles, France, Sept. 1995.
10. E. Keogh and P. Smyth. A Probablistic Approach to Fast Pattern Matching in Time-Series Databases . In *Proc. 1997 Int. Conf. on Knowledge Discovery and Data Mining (KDD'97)*, Newport Beach, California, Aug. 1997.
11. H. Mannila, H. Toivonen, and A. I. Verkamo. Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery*, 1:259–289, 1997.
12. D. Rafiei. On similarity-based queries for time series data. In *Proc 1999 Int. Conf. on Data Engineering (ICDE'99)*, pages 410–417, Sydney, Australia, Mar. 1999.
13. B.-K. Yi, H. V. Jagadish, and C. Faloutsos. Efficient retrieval of similar time sequences under time warping. In *Proc. 1998 Int. Conf. Data Engineering (ICDE'98)*, pages 201–208, Orlando, FL, Feb. 1998.

COFE: A Scalable Method for Feature Extraction from Complex Objects

Gabriela Hristescu and Martin Farach-Colton

{hristesc,farach}@cs.rutgers.edu
Rutgers University, Department of Computer Science,
Piscataway, New Jersey 08854, USA

Abstract. Feature Extraction, also known as Multidimensional Scaling, is a basic primitive associated with indexing, clustering, nearest neighbor searching and visualization. We consider the problem of feature extraction when the data-points are complex and the distance evaluation function is very expensive to evaluate. Examples of expensive distance evaluations include those for computing the Hausdorff distance between polygons in a spatial database, or the edit distance between macromolecules in a DNA or protein database.
We propose COFE, a method for sparse feature extraction which is based on novel random non-linear projections. We evaluate COFE on real data and find that it performs very well in terms of quality of features extracted, number of distances evaluated, number of database scans performed and total run time. We further propose COFE-GR, which matches COFE in terms of distance evaluations and run-time, but outperforms it in terms of quality of features extracted.

1 Introduction

Feature Extraction, also known as Multidimensional Scaling (MDS), is a basic primitive associated with indexing, clustering, nearest neighbor searching and visualization. The simplest instance of feature extraction arises when the points of a data set are defined by a large number, k, of features. We say that such points are *embedded in k-dimensional space*. Picking out $k' \ll k$ features to represent the data points, while preserving distances between points, is a feature extraction problem called the *dimensionality reduction* problem.

The most straightforward and intuitively appealing way to reduce the number of dimensions is to pick some subset of size k' of the k initial dimensions. However, taking k' linear combinations of the original k dimensions can often produce substantially better features than this naïve approach. Such approaches are at the heart of methods like Single Value Decomposition (SVD) [10] or the Karhunen-Loève transform [8]. Linear combinations of original dimensions are but one way to pick features. Non-linear functions of features have the potential to give even better embeddings since they are more general than linear combinations.

In many other cases the points are complex objects which are not embedded. For example, if the dataset consists of DNA or protein sequences then there is no

Y. Kambayashi, M. Mohania, and A M. Tjoa (Eds.): DaWaK 2000, LNCS 1874, pp. 358–371, 2000.
© Springer-Verlag Berlin Heidelberg 2000

natural notion of a set, large or otherwise, of orthogonal features describing the objects. Similarly, in multimedia applications, the data may include polygons as part of the description of an object. While such data types are not represented in a feature space, they are typically endowed with a *distance function*, which together with the dataset of objects define a *distance* space. For example, the distance between biological macromolecules is taken to be some variant of the edit distance between them. For geometric objects in 2- or 3-dimensions, the distance is often measured as the Hausdorff distance. These distance functions are very expensive to compute. The edit distance between two sequences of length m takes $O(m^2)$ time to compute, while the Hausdorff distance between two geometric objects, each with m points, takes time $O(m^5)$ to compute. Even though complex objects have a finite representation in the computer, the natural feature space this representation describes does not preserve distances between objects. For example, the k points of the polygon can be trivially represented by $O(k)$ dimensions by a straightforward $O(k)$ bit computer representation, but the vector distance between such embeddings is unrelated to any natural geometric distance between the polygons.

The *Complex Object Multidimensional Scaling (COMDS)* problem is then the problem of extracting features from objects given an expensive distance function between them. A good solution to the *COMDS* problem has to have good quality:
Quality: The k features extracted should reflect, as closely as possible, the underlying distances between the objects. Furthermore, the extracted features should be good even with small k. If we are interested in visualization $k = 2, 3$. Clustering and nearest neighbor searching becomes prohibitively expensive, or the quality of the clustering degrades, if k is more than about 10. Thus the quality of a *COMDS* algorithm depends on the quality of a small number of extracted features. There is a tradeoff between the quality and the scalability of a solution to the *COMDS* problem.
A scalable solution should have the following characteristics:
Sparsity: Since the distance function is very expensive to evaluate, it is not feasible to compute all $\binom{n}{2}$ pairwise distances, where n is the number of elements in the database. Thus, the method must compute only a sparse subset of all possible distances.
Locality: As many databases continue to grow faster than memory capacity, the performance of any *COMDS* solution will ultimately depend on the number of accesses to secondary storage. Therefore, a *COMDS* solution should have good locality of object references. This can be measured by the number of database scans necessary to compute the embedding. Thus, this number should be low.

We address these issues in designing the (*Complex Object Feature Extraction*) (COFE) method, an algorithm for the *COMDS* problem. We evaluate COFE and compare it with FastMap [7], a previously proposed solution for *COMDS*.

Features define a metric space. The standard way to define the distance between two k-dimensional feature vectors is through their l_2 (Euclidean) distance, that is, if point p has features $p_1, ..., p_k$ and point q has features $q_1, ..., q_k$ we can interpret the "feature distance" $d'(p, q) = \sqrt{\sum_{i=1}^{k} (p_i - q_i)^2}$. Taken in this

view, we seek the embedding of the real distance function $d(\cdot,\cdot)$ into $\Re^k$, such that the induced l_2 distance function $d'(\cdot,\cdot)$ is as similar to $d(\cdot,\cdot)$ as possible. We chose to use the Euclidean distance between feature vectors because a considerable amount of database work has focused on indexing and related tasks for low dimensional Euclidean data. No other metric combines the robustness of a low dimensional Euclidean space with its tractability. A survey of techniques which can be applied to low-dimensional Euclidean data can be found in [7].

Measuring the Quality of an Embedding. We compared embeddings based on the *Stress* criterion proposed in [13]. The *Stress* of a distance function $d(\cdot,\cdot)$ with respect to another distance function $d'(\cdot,\cdot)$ is defined as:

$$Stress(d,d') = \sqrt{\frac{\sum_{i,j}(d(i,j)-d'(i,j))^2}{\sum_{i,j}d'(i,j)^2}}.$$

2 Related Work

Multidimensional Scaling (MDS) has been used in the literature with different meanings, though in this paper we restrict ourselves to the standard meaning within the database community [14], as described above. MDS can be applied as a preprocessing step for data to be used in indexing, clustering and related tasks. The method we propose, called COFE, is a scalable MDS method.

Traditional MDS methods [15] do not offer a scalable solution to the *COMDS* problem, because of high complexity and unrealistic resource assumptions. In [7], Faloutsos and Lin proposed the FastMap method for MDS. Their innovation was to consider sparsity as an important factor for MDS and FastMap is the first method we know of which maintains quality and sparsity simultaneously. However, Faloutsos and Lin did not consider the problem of scalability – the largest instance which they used to evaluate their method had 154 points, consisted of 13-dimensional categorical data – and FastMap is not optimized for complex objects. Faloutsos and Lin [7] discuss traditional MDS methods [15], their drawbacks and compare them with FastMap. We describe FastMap in detail below, and make extensive comparisons between FastMap and COFE.

Once an MDS method has embedded the database in low-dimensional Euclidean space, efficient methods for indexing or clustering, to support nearest neighbor search or visualization, can be applied. A considerable amount of database work has focused on indexing, clustering and related tasks for low dimensional Euclidean data. Here, we mention briefly some of the key results in this area, in particular for the problem of *similarity querying*, which has been used to mean nearest neighbor searching in a database.

Multi-dimensional index structures called Spatial Access Methods (*SAMs*) were designed to rely on certain clustering properties of objects in low-dimensional space. If these assumptions hold we would expect objects to occupy regions of the multi-dimensional space that can be captured by the multi-dimensional index. The result of previous research [3, 4, 20] indicated that the indexing techniques, which are designed for low-dimensional data, do not perform equally well on

high-dimensional data. Some of the most popular database indexing structures used for similarity querying are the R^*-tree [5], X-tree [4], SS-tree [20], SR-tree [12], TV-tree [16], the Pyramid Technique [2], PK-tree [21]. BUBBLE and BUBBLE-FM [9] are two recently proposed algorithms for clustering datasets in arbitrary distance spaces, based on BIRCH [22], a scalable clustering algorithm. BUBBLE-FM uses FastMap to improve scalability.

3 Embedding Methods

FastMap. The approach taken in FastMap [7] for embedding points into k-dimensional Euclidean space is based on 3-point linear projections. Consider three objects O_a, O_b and O_i, and the distances, $d(a, b)$, $d(a, i)$ and $d(b, i)$, between them. Any such distances which satisfy the triangle inequality can be represented exactly in two-dimensional Euclidean space (Figure 1). O_a can be assumed to be at position $(0, 0)$, and O_b at $(0, d(a, b))$. To find O_i's two coordinates (x_i, y_i), we can solve two equations with two unknowns, plugging in the values $d(a, i)$ and $d(b, i)$.

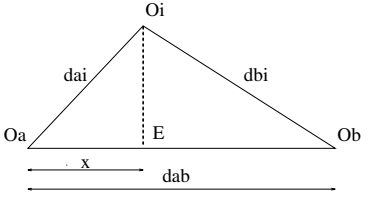


Fig. 1. Projecting point O_i on the line O_aO_b

Therefore, each feature is a linear projection defined by a reference set O_a and O_b. The question is then how to pick such a reference pair. Faloutsos and Lin's aim was to select a pair along the principal component, *a la* SVD. Since it would be computationally expensive to find the principal component of the points, they suggest finding the pair which is furthest apart, which they suggest should lie more or less along the principal component. But finding such pair is also computationally expensive. So the following heuristic is used: Start by picking a point O_a at random. Find the furthest point to O_a by computing its distance to all others and consider it O_b. Compute O_b's distance to all others and replace O_a with the most distant point. Repeat the last two steps t times, where t is a parameter of the algorithm. The last pair of points O_a and O_b found this way are taken to be the reference points. Note that this sketch only describes how the first feature is selected. All distance calculations for the second feature factor out the part of the distance function which has already been captured by the first feature. Otherwise, if O_a and O_b are the furthest points originally, they will remain so and all features will be identical. The selection of a reference set always requires t one-against-all sets of distance evaluations. Thus, selecting the reference points requires $O(tn)$ distance evaluations. Computing the feature once the reference set is known requires another $2n$ distance evaluations. Therefore, k dimensions require $O(tkn)$ distance evaluations.

FastMap-GR. Once a method has extracted k features, the dimensionality of the embedding space can be further reduced by selecting a smaller set k' of high

quality features from the initial k. The quality of a feature can be quantified in terms of how well it reduces the *Stress* [13]. We propose the *Greedy Resampling* algorithm for picking out the best subset of features, which has the advantage of sparsity over traditional methods like SVD [10], the Karhunen-Loève transform [8] and even the recent and more efficient SVD-based method [11].

FastMap-GR starts by picking out k features using FastMap. Then, each single feature is compared in terms of what *Stress* it produces. Notice that computing the *Stress* directly seems to require $\binom{n}{2}$ distance evaluations, where n is the size of the dataset, but we can simply pick out a random sample of distances and compare the *Stress* on just these distances. Thus, picking out the best feature is very fast compared to finding the k features to begin with. Once some feature f_1 is selected as the best, we can pick the second feature as the one which produces the best *Stress* when combined with f_1. We can proceed in this greedy manner until we have reordered all k features by decreasing order of quality.

Having done this, if we need $k' < k$ features, we can simply take the first k' features of the reordered embedding. Thus, *Greedy Resampling* is a dimensionality reduction method. Note that it does not guarantee to pick out the best k' features, but picking out the best k' features takes exponential time, and so we suggest greedy resampling as a heuristic.

Bourgain. Bourgain's method [6] is not a sparse method, that is, it evaluates all $\binom{n}{2}$ possible distances between the n objects of the database. We will therefore not compare its performance with those of the other methods presented in this section. However, COFE is based on the Bourgain embedding, so we present this method for the sake of exposition.

Suppose we have a set X of n elements. Let the distance function between elements of X be $d : X \times X \to \Re^+$. We define the distance function D between an element of X and a subset $X' \subseteq X$ as $D(x, X') = \min_{y \in X'}\{d(x, y)\}$, that is, $D(x, X')$ is the distance from x to its closest neighbor in X'. Let $R = \{X_1, X_2, ..., X_k\}$ be a set of subsets of X. Then we can define an embedding with respect to R as follows: $E_R(x) = [D(x, X_1), D(x, X_2), ..., D(x, X_k)]$. It is not obvious at first why such an embedding might have any reasonable properties. For one thing it is highly non-linear, and so visual intuition tends not to be too helpful in understanding the behavior of such an embedding. However, we can gain some understanding of this type of embedding by considering the case where the points of the metric space are packed tightly into well separated clusters. Then, if some set X_1 has a point which is in one cluster C_1 and not in another C_2, the distance $D(x, X_1)$ will be small for $x \in C_1$ and large for $x \in C_2$. The dimension corresponding to X_1 will then induce a large distance between points in C_1 and C_2. Obviously we cannot count on the input points being packed into well-separated clusters, or that the sets X_i have the right arrangements with respect to the clusters. However, Bourgain showed that there is a set of reference sets which works for general inputs.

The Bourgain embedding, then, is a choice of reference sets, R, and the corresponding embedding E_R. The choice of the sets in R will be randomized via a procedure described below: R consists of $O(\log^2 n)$ sets $X_{i,j}$, which we

can think of as naturally organized into *columns* (κ) and *rows* (β). Let $R = \{X_{1,1}, X_{1,2}, ..., X_{1,\kappa}, X_{2,1}, ..., X_{2,\kappa}, X_{\beta,1}, ..., X_{\beta,\kappa}\}$, where $\kappa = O(\log n)$ and $\beta = O(\log n)$. The Bourgain embedding will thus define $O(\log^2 n)$ dimensions. We select the elements of $X_{i,j}$ to be any random subset of X of size 2^i. Thus, we get κ sets of size 2, κ of size 4, etc., up to β of size approximately n. Here is an example of the Bourgain embedding. Let $d(\cdot,\cdot)$ be given by the following matrix:

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8
x_1	0	12	10	9	10	6	3	10
x_2		0	8	13	14	10	13	8
x_3			0	11	12	8	11	2
x_4				0	3	7	10	11
x_5					0	8	11	12
x_6						0	7	8
x_7							0	11
x_8								0

The algorithm picks 6 reference sets at random, 3 with 2 elements, and 3 with 4 elements:

x_3, x_4	x_1, x_8	x_2, x_6
x_4, x_5, x_2, x_3	x_1, x_4, x_6, x_3	x_7, x_5, x_3, x_6

The embedding of x_3 and x_5 is computed as:

$$E_R(x_3) = \begin{array}{|c|c|c|}\hline 0 & 2 & 8 \\ \hline 0 & 0 & 0 \\ \hline \end{array} \quad \text{and} \quad E_R(x_5) = \begin{array}{|c|c|c|}\hline 3 & 10 & 8 \\ \hline 0 & 3 & 0 \\ \hline \end{array}$$

The distance between x_3 and x_5, originally 12 is: $l_2(E_R(x_3), E_R(x_5)) = \sqrt{3^2+8^2+0+0+3^2+0}$.

An excellent presentation of the proof of the correctness for the Bourgain embedding can be found in [17]. Bourgain showed [6] that if R is selected as above, then the embedding E_R has *distortion* $O(\log n)$, where the distortion of an embedding is the maximum stretch of any distance, that is $d(x_i, x_j) \leq \log n l_2(E_R(x_i), E_R(x_j))$, and $d(x_i, x_j) \geq l_2(E_R(x_i), E_R(x_j))/\log n$.

At first glance it seems that modifying a distance by as much as a $\log n$ *multiplicative* factor would almost completely destroy the original information contained in the distance function $d(\cdot,\cdot)$. For example, if the we are embedding 1024 points, and $d(\cdot,\cdot)$ ranges over a factor of 10, a $\log n$ distortion embedding could arbitrarily reorder the distance between objects in our dataset. However, as with many worst case bounds, the Bourgain guarantee of no more than $\log n$ distortion is often very far from the actual behavior on real data. Furthermore, the $\log n$ bound is tight in the sense that there are metrics which require this much distortion for any Euclidean embedding. Experimental analysis on meaningful data rather than on concocted counterexamples is the *sine qua non* of feature extraction evaluation.

The Bourgain embedding has two very serious drawbacks. First, it produces $O(\log^2 n)$ dimensions. Again, if we are embedding 1024 points, we could end up with 100 dimensions, which would be far too many. Second, with high probability, every point in the dataset will be selected in some reference set $X_{i,j}$. Thus, we will need to compute all $\binom{n}{2}$ distances in order to perform the Bourgain embedding. If we are using the embedding for similarity searching, we would need to compute $E_R(q)$ for a query point q. This would require a distance evaluation between q and every point in X. There would be no point in performing an embedding if we were willing to evaluate all n distances between q and the points in X, since we are using the embedding in order to facilitate a nearest neighbor search.

Similarly, the Karhunen-Loève transform is completely impractical. FastMap was, in fact, designed to give a practical alternative to the Karhunen-Loève transform.

COFE. COFE is a heuristic modification of the Bourgain method which is designed to perform very few distance evaluations. The key idea in COFE is a loop interchange. Bourgain's method is typically thought of as having an outer loop which runs through all the points, and then an inner loop which computes all the features. We can equivalently imagine computing the first feature for every point, then the second feature, etc., until we have computed however many features we want. So far, we have not changed anything substantial: whether we loop on all points and then on all reference sets, or on all reference sets and then on all points, we end up doing the same distance evaluations. Notice that the features are defined by a two-dimensional array. Thus, the order in which we loop through the features is not determined. For reasons which are suggested by the proof of the distortion bound of Bourgain's algorithm, we will proceed row-by-row, that is, we will compute features for all reference sets of size 2, then for size 4, etc.

So finally, we need to know how to actually save on distance evaluations. The main idea is as follows: Suppose we already computed k' features for every point. Then we have a rough estimate on the distances between points, that is $d'_{k'}(p,q) = \sqrt{\sum_{l=1}^{k'}(p_l - q_l)^2}$, where p_l is the l^{th} feature of p, and similarly for q. If the first k' features are good enough, then we can very quickly compute the approximate distance from some point to all points in a reference set, using these first k' features. We can then consider just the best few candidates, and perform full distance evaluations to these points. Concretely, for every point p in the dataset X and every reference set X_k (taken in row-major order) the approximate distance $\tilde{D}(p, X_k)$ is computed as follows:

- For every point $q \in X_k$, compute the approximate distance $d'_{k-1}(p,q)$.
- Select the σ points in X_k with smallest d' distance to p.
- For each point q of these σ points, evaluate the true distance $d(p,q)$.
- Then $\tilde{D}(p, X_k) = d(p, q')$, where q' is the q with smallest $d(p,q)$.

$\tilde{D}(p, X_k) = \min_{y \in S}\{d(p,y) | S \subseteq X_k,\ |S| = \sigma, \forall a \in S, \forall b \in X_k \setminus S, d'(p,a) \leq d'(p,b)\}$
Thus, for every point, and for every feature, we compute σ distances. We get $O(nk\sigma)$ distance evaluations, if there are n points and k features.

There are two questions one might ask about the COFE embedding: does it do well in theory and does it do well in practice. The second question, which is the subject of this paper, is addressed in Section 4. As far as the theory goes, COFE does not provide the distortion guarantees of Bourgain. Indeed, we have shown that *no* algorithm which evaluates any proper subset of the values of $d(\cdot,\cdot)$ can give such a guarantee. However, as noted above, a guarantee of $\log n$ distortion is not really all that useful, so we turn our attention to the matter of evaluating the performance of COFE on data.

COFE-GR. Recall FastMap-GR, which uses FastMap to find k features and then reorders them using the Greedy Resampling heuristic. COFE-GR is analogous: it uses COFE to find k features and then uses Greedy Resampling to reorder the features. We will show that COFE-GR does a very good job of picking out the best features from a COFE embedding.

4 Experimental Comparison

We compared COFE, COFE-GR, FastMap and FastMap-GR in terms of the quality of the computed embedding, the execution time for computing the embedding and the number of distance evaluations performed during the computation.

4.1 Protein datasets

We chose proteins as test data, because the distance function between them is very expensive to compute. If proteins are compared at the structure level, just one distance computation can take minutes and even hours. Comparing proteins at sequence level is faster, but still computationally high, if no specialized hardware is used. We ran experiments on datasets selected randomly from the SwissProt protein database [1]. In [7], only very low dimensional data was used, such as 3 to 13 dimensional vector data, with the exception of text data. However, computing the distance between text documents, once such preprocessing as finding word-counts has been completed, is not expensive and thus not as relevant to our method.

The size of the protein datasets we used ranged from 48 to the whole SwissProt database consisting of 66,745 proteins. The platform we used for these experiments was a dual-processor 300 MHz PentiumII with 512 MB of memory. COFE and COFE-GR were implemented in C and tested under Linux. We used the original FastMap code supplied by Faloutsos and Lin [7]. Due to excessive memory requirements we were not able to run FastMap on datasets larger than 511. Thus, most of our comparisons will be the average of 10 random subsets of size 255 from SwissProt. We find the performance of FastMap and COFE not to degrade significantly with size, and so these data are quite representative.

We used the Smith-Waterman [19] measure for protein similarity, which is based on a dynamic programming method that takes $O(m_1 m_2)$ time, where m_1 and m_2 are the lengths of the two protein sequences. Since this is a very expensive operation, it will significantly affect the running time of the methods that use it. If $s(a, b)$ is the similarity value between proteins a and b, the distance measure between the two proteins we considered is $d(a, b) = s(a, a) + s(b, b) - 2s(a, b)$ as proposed by Linial & al. [18].

4.2 The Quality of Embeddings

For the quality of the embedding we used the *Stress* measure suggested by Faloutsos et al. [7] which gives the relative error the embedded distances suffer from on average. The figures report results for embeddings of sizes ranging from one to the maximum number of features considered. We used $t = 2$ for FastMap.

Notice that in COFE we can select a number of rows (α) and columns (κ) to evaluate. The number of features extracted will then be $k = \alpha\kappa$. We used the full Bourgain dimensionality with $\kappa = \alpha = \log n$ and $\sigma = 1$ for COFE.

We measured the *Stress* on 4,000 distances between randomly selected pairs of elements from the dataset. In our comparisons we present the average over results on 10 sets of 255 proteins.

We begin by comparing COFE with FastMap and COFE-GR with FastMap-GR (Figure 2). Results show that for FastMap the *Stress* starts at 0.88 for

the first feature extracted and constantly decreases with increasing number of features, reaching 0.37 for 49 features, which constitutes a 58% reduction. For the same range of features, COFE starts with a much lower *Stress* value (0.54) and ends at 0.33, although the *Stress* between the two methods after 14 features is basically indistinguishable. Comparing COFE-GR with FastMap-GR we notice the quality difference between them. COFE-GR starts by being 46% better than FastMap for the first extracted feature and stays around 50% for the first 10 features, after which the gap in quality decreases to almost 0 for 46 features.

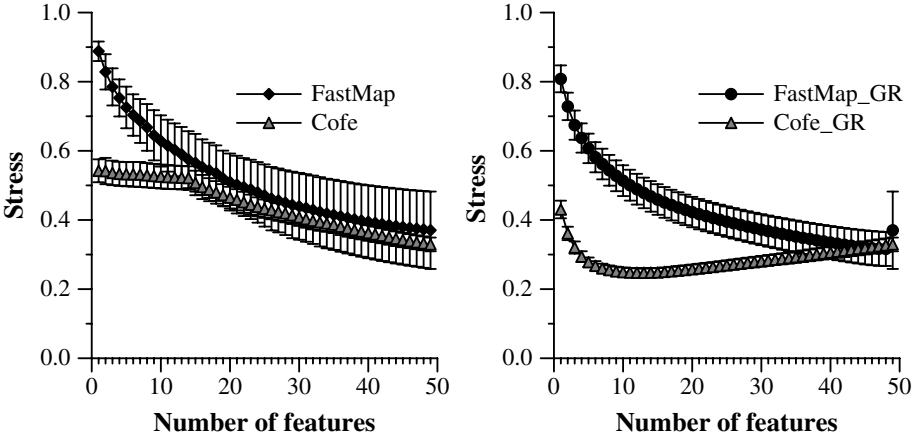


Fig. 2. Average *Stress* of COFE vs. FastMap and COFE-GR vs. FastMap-GR on 10 sets of 255 proteins (with error bars)

The thing to note is that COFE-GR has its best *Stress* at 10 dimensions, and this *Stress* is not matched by FastMap-GR, even out to 49 dimensions (though if sufficient dimensions were used, FastMap would probably produce a lower-*Stress* embedding). As noted above, it is quality of an embedding with a small number of dimensions which is most important, and so we conclude that COFE-GR produces very high quality embeddings with a small number of features.

The Effects of Greedy Resampling. Intuitively, the greedy resampled versions of the algorithms should have lower *Stress* for any given number of features, unless, of course, we use all features, in which case the greedy resampled algorithms will match their non-resampled versions. The comparison of the two plots in Figure 2 confirms this, although we note that greedy resampling is a heuristic and that it is possible to come up with pathological embeddings where greedy resampling does not yield good reduction of dimensionality. A striking aspect of the results is that COFE-GR is a significant improvement over COFE, while FastMap and FastMap-GR have roughly similar performance. COFE-GR starts by improving the quality of the first selected feature by 20% over COFE and reaches the minimum *Stress* of 0.25 for 10 features, whereas FastMap-GR starts with a 9% improvement over FastMap and reaches the minimum value of 0.31 for 46 features. This comparison shows that COFE can significantly benefit from resampling, while FastMap selects the sequence of choices of reference sets very close to the one picked through greedy resampling on the same reference set.

4.3 The Execution Time

In addition to comparing the quality of the two methods, we were also interested in comparing their execution time. We instrumented the code to get timing

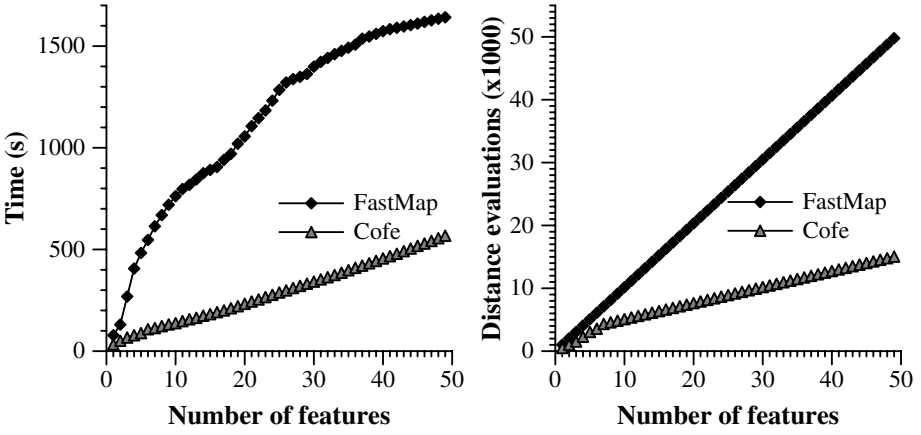


Fig. 3. The average run time and the number of distance evaluations for COFE vs. FastMap for 255 proteins

information and also profiled the execution in order to understand where the time was spent. Recall that greedy resampling does not require a significant number of distance evaluations and very little run time, so we compare only COFE with FastMap.

The first plot in figure 3 presents the average execution time over runs on 10 datasets of size 255. It shows results for increasing number of features for COFE and FastMap. COFE outperforms FastMap by being 2.3 times faster than it for the first feature and 2.8 times faster for all 49 features.

We profiled both methods to determine where the bottlenecks were in the computation. As expected, in both cases, more than 90% of the execution time was spent in the computation of the distances between data points (proteins). The difference in execution time is explained by the difference in the frequency of distance computations between the two methods. However, the run times measured require some explanation. We would have expected FastMap to run about 33% slower than COFE since it performs about 4/3 as many distance evaluations in theory. However, two effects come into play here. First, FastMap performs more scans of the data, and we would expect it to pay a penalty in terms of slower memory access. Also, the reference set is known ahead of time in COFE. This allows many optimizations, both in terms of possible parallelization, as well as in savings of repeated distance evaluations. Such optimizations are not possible for FastMap because the reference sets are only determined dynamically. Thus, Figure 3 presents actual measured data of optimized code.

In Figure 3 we also compare the number of distance calls for a dataset of 255 protein sequences. For a 49 feature embedding 46% of the total number of pairwise distances were computed for COFE and 153% for FastMap. Even for

one feature FastMap computes 3.1% of distances compared to 1.5% for COFE. Even if we use the lowest value for $t = 1$, which means that one reference point is completely random, FastMap makes 115% of the total number of pairwise distance evaluations for 49 features extracted and 2.3% for the first feature.

4.4 Scalability

We studied how the COFE method scales with increasing dataset sizes both in terms of quality and running time. The first plot in figure 4 shows the quality of the first 15 dimensions of COFE-GR. We note very consistent behavior, with basically no degradation in the solution over a wide range of dataset sizes. In figure 4 we also show the measured run time of COFE over a variety of data set sizes and number of features extracted. The flat initial part of the curve is due to startup costs.

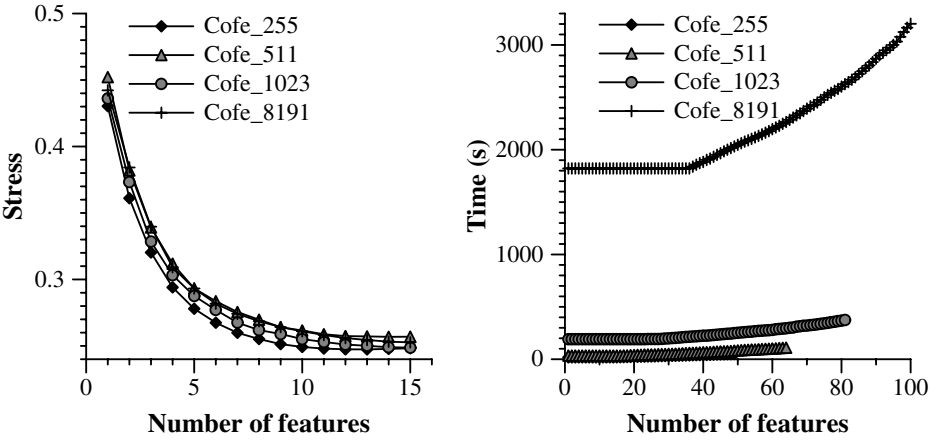


Fig. 4. Quality of COFE-GR features and run time for COFE for protein sets of sizes ranging from 255 to 8191

5 Conclusions

We have proposed two fast algorithms, COFE and COFE-GR which map points into k-dimensional space so that distances are well preserved. We assume only that we have a distance function between pairs of data-points. Such a setting is very natural for data types such as biological sequences and multimedia data. Since distance evaluations are very expensive, reducing the number of distance evaluations, while preserving the quality of our embedding was our main goal.

We compared our COFE method with FastMap [7] on 10 protein datasets, each of size 255. We found that our method performs substantially fewer distance evaluations than FastMap. Furthermore, the *Stress* of the embedding is about half that of FastMap's for a reasonable number of dimensions. We conclude that COFE is an extremely effective solution to the *Complex Object Multidimensional Scaling* problem, exhibiting excellent scalability and yielding high quality features.

6 Acknowledgements

This work was supported by NSF Award CCR-9820879. The authors thank Christos Faloutsos and King-Ip Lin for providing the source code of the FastMap method.

References

1. Bairoch,A., Apweiler, R.: The SWISS-PROT protein sequence data bank and its supplement TrEMBL in 1998. Nucleic Acids Res. **26** (1998) 38–42
2. Berchtold, S., Bőhm, C.: The Pyramid-Technique: Towards breaking the curse of dimensionality. Proc. ACM SIGMOD Conf. (1998) 142–176
3. Berchtold, S., Bőhm, C., Keim, D.A., Kriegel, H.-P.: A cost model for nearest neighbor search in high-dimensional data space. Proc. ACM PODS Symposium (1997)
4. Berchtold, S., Keim, D.A., Kriegel, H.-P.: The X-tree: An index structure for high-dimensional data. Proc. 22nd VLDB Conf. (1996) 28–39
5. Beckmann, N., Kriegel, H.-P., Schneider, R., Seeger, B.: The R^* tree: An Efficient and Robust Access Method for Points and Rectangles. Proc. ACM SIGMOD Conf. (1990) 322–331
6. Bourgain, J.: On Lipschitz embedding of finite metric spaces in Hilbert space. Israel J. of Math. (1985) 52:46–52
7. Faloutsos, C., Lin, K.-I.: FastMap: a fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. Proc. ACM SIGMOD **24(2)** (1995) 163–174
8. Fukunaga, K.: Introduction to Statistical Pattern Recognition. Academic Press, 2nd Edition (1990)
9. Ganti, V., Ramakrishnan, R., Gehrke, J., Powell, A., French, J.: Clustering large datasets in arbitrary metric spaces. Proc. 15th ICDE Conf. (1999) 502–511
10. Golub, G. H., Van Loan, C. F.: Matrix computations. Johns Hopkins University Press, 2nd Edition (1989)
11. Kanth, K. V. R., Agrawal, D., Singh, A.: Dimensionality reduction for similarity searching in dynamic databases. Proc. ACM SIGMOD Conf. (1998) 142–176
12. Katayama, N., Satoh,S.: The SR-tree: An index structure for high-dimensional nearest neighbor queries. Proc. ACM SIGMOD Conf. (1997) 369–380
13. Kruskal, J.B.: Multidimensional Scaling by Optimizing Goodness of Fit to a Nonmetric Hypothesis. Psychometrika **29** (1964) 1–27
14. Kruskal, J.B.: Multidimensional Scaling and other Methods for Discovering Structure. Stat. Meth. for Digital Computers, Wiley, New York (1977) 296–339
15. Kruskal, J.B., Wish, M.: Multidimensional Scaling. Sage University Paper series on Quantitative Applications in the Social Sciences, Beverly Hills, CA (1978) 7–11
16. Lin, K.-I., Jagadish, H. V., Faloutsos, C.: The TV-tree: An index structure for high-dimensional data. Proc. 20th VLDB Conf. **3(4)** (1994) 517–542
17. Linial, N., London, E., Rabinovich, Y.: The geometry of graphs and some of its algorithmic applications. Proc. 35th IEEE FOCS Symp. (1994) 577–591
18. Linial, M., Linial, N., Tishby, N., Yona, G.: Global self organization of all known protein sequences reveals inherent biological signatures. J. Mol. Biol. **268** (1997) 539–556
19. Smith, T., Waterman, M.: The identification of common molecular subsequences. J. Mol. Biol. **147** (1981) 195–197

20. White, D. A., Jain, R.: Similarity Indexing with the SS-tree. Proc. 12th ICDE Conf. (1996) 516–523
21. Wang, W., Yang, J., Muntz, R. R.: PK-tree: A Spatial Index Structure for High Dimensional Point Data. 5th Intl. FODO Conf. (1998)
22. Zhang, T., Ramkarishnan, R., Livny, M.: Birch: An efficient data clustering method for large databases. Proc. ACM SIGMOD Conf. (1996) 103–114

A Analytical Comparison

In this Appendix we compare COFE, COFE-GR, FastMap and FastMap-GR in terms of the following three criteria: quality, sparsity and locality.

Quality. As stated in Section 3, COFE is based on the Bourgain algorithm, which guarantees a distortion of $O(\log n)$ for any metric, and there are some metrics which require $\Omega(\log n)$ distortion. It is easy to show that no sparse method can have such a bound. Consider, for example, the *needle-in-a-haystack* distance function. All distances are some large Δ, except, perhaps, for some unknown pair a, b which are at distance $\epsilon \ll \Delta$ apart. Unless we find the pair a, b, we do not know on any unevaluated distance if it is Δ or ϵ. Thus we cannot provide an embedding with distortion of less than Δ/ϵ, a quantity which is not even bounded. But this is not a shortcoming of COFE, it is a shortcoming of *every sparse method*, including FastMap.

Similarly, FastMap is based on the Karhunen-Loève transform, which gives some optimality guarantees about picking out the best *linear projections.* Notice that it gives no guarantees on finding the best features overall, only the best linear features, under some criterion. FastMap is a heuristic and gives no guarantees. We conclude by noting that there is no sparse method which gives a guarantee under any criterion.

Sparsity. Above, we showed that FastMap performs $O(tkn)$ distance evaluations, where t is a parameter of the algorithm and COFE performs $O(\sigma kn)$ evaluations. The greedy resampling versions of these algorithms do not perform any significant number of extra distance evaluations.

A comparison of the exact number of distance evaluations performed requires knowing t and σ, and analyzing the hidden constants in the O notation. If we consider a dataset with n elements, the average computational cost of a distance evaluation in the distance space to be D and the cost of computing the Euclidean distance in an i-dimensional space to be $d_i = id$ where d is the cost of computing a squared difference. We consider the number of features extracted to be k.

FastMap first chooses a pair of points that define a reference line and then projects all points in the dataset on that line. At the next step, this procedure is repeated, except that the distances between points are considered projected in a hyperplane orthogonal to the chosen reference line. This introduces, in addition to a distance evaluation in the distance space, an extra d computation for each feature already extracted. The time to execute FastMap consists of the time $T_{extract}$ to extract the k features and the time $T_{project}$ to project the n points onto them. If for each feature extracted, a number of t passes over the dataset are executed while on each pass the furthest point to a selected reference point is computed, then $T_{extract} = t \sum_{i=1}^{n} \sum_{j=1}^{k} (D + (j-1)d)$ and if c distance computations are needed to project a point onto a reference line, then $T_{embed} = c \sum_{i=1}^{n} \sum_{j=1}^{k} (D + (j-1)d)$. The number of distance computations c for projecting a point on a line is 2, considering that the distance between the reference points is known. Thus: $T_{FastMap} = T_{extract} + T_{embed} = nk(t+c)\left(D + \frac{k-1}{2}d\right)$.

Computing the Bourgain transform would require randomly selecting the reference sets and then computing the embedding of a point in the image space as the smallest

distance between the point and the points in the reference sets. Selecting the reference sets does not require any computation, and therefore for an embedding with κ columns and α rows: $T_{Bourgain} = 2n\kappa(2^{\alpha} - 1)D$.

Instead of the α rows Bourgain would compute, COFE computes only β rows using the distance in the distance space. For the rest of $\alpha - \beta$ rows, the coordinate for the corresponding dimension i is approximated by first selecting the σ closest points in the already computed $(i-1)$-dimensional image space, then computing the smallest distance in the distance space among the σ points.

Extending the β row embedding to an α row embedding takes therefore: $T_{extension} = n\kappa\sigma(\alpha - \beta)D + n\sum_{i=\beta+1}^{\alpha}\left(2^i \sum_{j=\kappa(i-1)}^{\kappa i-1}(jd)\right)$. Approximating $j \leq \kappa i$ we get: $T_{Cofe} \leq n\kappa\left(2^{\beta+1} - 2 + \sigma(\alpha-\beta)\right) D + n\kappa\left((\alpha-1)2^{\alpha+1} - \beta 2^{\beta+2} + 4\right) d$.

Because computing D is by far more expensive than d, the dominant terms for the two methods are: $T'_{FastMap} = n\kappa\alpha(t+c)D$ and $T'_{Cofe} = n\kappa\left(2^{\beta+1} + \sigma(\alpha-\beta)\right)D$, where the size of the embedding is $k = \kappa\alpha$.

For $\alpha = logn$ and $\beta = loglogn$:
$T'_{FastMap} = n\kappa(t+c)lognD$ and $T'_{Cofe} = n\kappa((\sigma+2)logn - \sigma loglogn)D$. For this choice of parameters, both methods compute $O(logn)$ real distances. Considering $c = 2$, and $k = \kappa logn$, FastMap performs more or less $(t+2)kn$ distance evaluations and COFE performs approximately $(\sigma+2)kn$ distance evaluations. The values for t that would give comparable numbers of distance computations in the distance space are $t = \sigma$ or $t = \sigma + 1$.

Faloutsos and Lin suggested that $t = 7$, though experiments showed that on the data used, very little advantage is gained by setting $t > 2$. We found that setting $\sigma = 1$ gave very reasonable result, so we set $t = 2$ and conclude that, as n increases, FastMap will perform $\frac{t+2}{\sigma+2} = \frac{4}{3}$ times as many distance evaluations.

Locality. In order to select one feature, FastMap has to scan the data t times and then compute the embedding (projection) using one more scan. For extracting k features, the number of scans is: $S_{FastMap} = (t+1)k$.

COFE randomly selects the reference sets that correspond to features. We distinguish two cases: either the reference sets fit in memory or they don't. Consider the second case in which not all reference sets fit in memory. The size of the reference sets grows exponentially with the row index of the feature. The smallest sets can be considered to fit in memory. Since the only information needed to compute the embedding corresponding to the bootstrapping features are the points in the reference sets, two scans, one to load them and one to compute the embedding for each point in the dataset, are needed. To compute the embedding of a point for each extension feature, the embedding for the previous features for that point and for the reference set points are needed. This can be achieved by loading a few reference sets during a scan and then computing their embedding, followed by another scan to compute the embedding of the rest of the points in the dataset. The reference sets for the next feature can also be prefetched during the computation of the embedding for the current feature. The worst case number of scans is therefore: $S_{Cofe} = k + 1$.

When all points in the reference sets fit in memory, that is, when we compute sufficiently few features so that we need not generate very large reference sets, a scan for loading the reference set and the computation of their embedding followed by a scan during which the whole embedding for each point in the dataset is computed suffices. Only 2 scans are required in this case.

In the worst case and setting $t = 2$, FastMap performs 3 times as many scans.

The Pruning Power: Theory and Heuristics for Mining Databases with Multiple *k*-Nearest-Neighbor Queries

Christian Böhm, Bernhard Braunmüller, Hans-Peter Kriegel

University of Munich, Oettingenstr. 67, D-80538 Munich, Germany
{boehm,braunmue,kriegel}@informatik.uni-muenchen.de

Abstract. Numerous data mining algorithms rely heavily on similarity queries. Although many or even all of the performed queries do not depend on each other, the algorithms process them in a sequential way. Recently, a novel technique for efficiently processing *multiple similarity queries* issued simultaneously has been introduced. It was shown that multiple similarity queries substantially speed-up query intensive data mining applications. For the important case of multiple *k*-nearest neighbor queries on top of a multidimensional index structure the problem of scheduling directory pages and data pages arises. This aspect has not been addressed so far. In this paper, we derive the theoretic foundation of this scheduling problem. Additionally, we propose several scheduling algorithms based on our theoretical results. In our experimental evaluation, we show that considering the maximum priority of pages clearly outperforms other scheduling approaches.

1. Introduction

Data mining is a core information technology and has been defined as the major step in the process of Knowledge Discovery in Databases [9]. Many data mining algorithms are query intensive, i.e. numerous queries are initiated on the underlying database system. In the prominent case of multidimensional data spaces, similarity queries, particularly *k-nearest neighbor (k-nn) queries*, are the most important query type [17]. The process of finding the *k*-nearest neighbors of a given query object is a CPU and I/O intensive task and the conventional approach to address this problem is to use some multidimensional index structure [16], [10].

While several sophisticated solutions have been proposed for single *k*-nn queries, the problem of efficiently processing multiple *k*-nn queries issued simultaneously is relatively untouched. However, there are many applications where *k*-nn queries emerge simultaneously. The proximity analysis algorithm proposed in [13], for instance, performs a *k*-nn query for each of the "top-*k*" neighbor objects of any identified cluster in the database. The outlier identification algorithm proposed in [6] performs a *k*-nn query with a high value of *k* for each database object. Another example is the classification of a set of objects which can efficiently be done by using a *k*-nn query for each unclassified object [14]. In all these algorithms *k*-nn queries are processed sequentially, i.e. one after another. However, since many or even all queries do not depend on each other, they can easily be performed simultaneously which offers much more potential for query optimization. In [2], a novel technique for simultaneous query processing called *multiple similarity queries* has been introduced. The authors present a syntactical transformation of a large class of data mining algorithms into a form which uses multiple similarity queries. For the efficient processing of the transformed algorithms, the authors propose to

Y. Kambayashi, M. Mohania, and A M. Tjoa (Eds.): DaWaK 2000, LNCS 1874, pp. 372–381, 2000.
© Springer-Verlag Berlin Heidelberg 2000

load a page once and immediately process it for all queries which consider this page as a candidate page. It has been shown that this results in a significant reduction of the total I/O cost. By applying the triangle inequality in order to avoid expensive distance calculations, further substantial performance improvements can be achieved. If the underlying access method is scan-based, this scheme is straightforward. However, when performing multiple k-nn queries on top of a multidimensional index structure, the important question emerges in which order the pages should be loaded. This problem has not been addressed in [2]. In this paper, we study the theoretical background of this scheduling problem. In particular, by developing a stochastic model, we find the expected distance to a nearest neighbor candidate located in a considered page to be the key information in order to solve the scheduling problem. Then we propose and evaluate several scheduling techniques which are based on our theoretical results.

The rest of this paper is organized as follows. In section 2, we describe the general processing of multiple k-nn queries. In section 3, we study the theoretical background of the scheduling problem, and in section 4, we propose several scheduling techniques. The experimental evaluation is presented in section 5, and section 6 concludes the paper.

2. Multiple k-Nearest Neighbor Queries

We shortly describe the algorithm for processing multiple k-nn queries (cf. Fig. 1) which is based on the HS single k-nn algorithm proposed in [12]. The algorithm starts with m query specifications each consisting of a query point q_i and an integer k_i. For each query an active page list (APL) is created. An APL is a priority queue containing the index pages in ascending order of the minimum distance MINDIST between the corresponding page regions and the query point. In contrast to single query processing, the multiple k-nn algorithm maintains not one APL at a time but m APLs simultaneously. While at least one k-nn query is running, the algorithm iteratively chooses a page P and each query q_i which has not pruned P from its APL_i (i.e. P is still enlisted in APL_i or P was not yet encountered) processes this page immediately. Thus, we only have *one* loading operation even if all queries process P. When processing a page P, the algorithm additionally saves valuable CPU time by avoiding distance calculations which are the most expensive CPU operations in the context of k-nn search. The method process(P, q_j) does not directly calculate the distances $d(q_j, o_s)$ between objects o_s located on P and a query object q_j, but instead, it first tries to disqualify as many objects o_s as possible by applying the triangle inequality in the following way: if $|d(q_i, q_j) - d(q_i, o_s)| \geq \text{knn_dist}(q_j)$, $i < j$, holds, then $d(q_j, o_s) \geq \text{knn_dist}(q_j)$ is true and the distance $d(q_j, o_s)$ needs not to be calculated. The inter-query distances $d(q_i, q_j)$ are precalculated and knn_dist(q_j) denotes the k-nn distance of query q_j determined so far. For all objects which cannot be disqualified, the distance to q_j is computed and stored in the buffer dist_buffer. Note that disqualified distances cannot be used to avoid distance computations of other query objects. Obviously, the method choose_candidate_page() is crucial for the overall performance of the algorithm, since a bad choice results in unnecessary and expensive processing of pages. Thus, it is important to find a schedule which minimizes the total number of processing operations.

```
DB::multiple_knn_queries(q_1, ..., q_m: query_object,
                                 k_1, ..., k_m: integer)
begin
   precalculate_interquery_distances([q_1, ..., q_m]);
   for i from 1 to m do create_APL(q_i);
   while queries_running([q_1, ..., q_m]) do
      P = choose_candidate_page([APL_1, ..., APL_m]);
      for i from 1 to m do APL_i.delete(P);
      load(P);
      initialize_dist_buffer();
      for i from 1 to m do
         if isa_candidate_page(P, q_i) then
            process(P, q_i, ...);
   return ([kNN_1, ..., kNN_m]);
end.

DB::process(page P, query_object q_i, ...)
begin
   foreach object o in P do
      if not avoid_dist_computation(o, q_i, dist_buffer) then
         dist_buffer[o][q_i] = distance(o, q_i);
         if dist_buffer[o][q_i] ≤ knn_dist(q_i) then
            if isa_directory_page(P) then APL_i.insert(o);
            else (* P is a data page *)
               kNN_i.insert(o);
               if kNN_i.cardinality() > k_i then
                  kNN_i.remove_last_object();
                  APL_i.prune_pages();
   return;
end.
```

Fig. 1. Multiple *k*-nearest neighbor algorithm

3. The Pruning Power Theory

The HS algorithm for processing single *k*-nn queries loads exactly those pages which are intersected by the *k*-nn sphere in ascending order of the MINDIST. In its basic form, the HS algorithm is heavily I/O bound. This behavior changes if we are moving from processing single *k*-nn queries to processing multiple *k*-nn queries. To explain this fact, we have to consider the loading operation load(P) and the processing operation process(P, q_i). When P is a data page, the processing operation determines the distances between the query point q_i and all data points on page P which cannot be avoided by applying the triangle inequality (cf. section 2). In the worst case, the total number of distance computations to be performed for a page P with capacity C_{eff} is $m \cdot C_{\text{eff}}$, i.e. no distance computation could be avoided. The cost for loading P, on the other hand, is independent of the number of queries which actually process this page and therefore remains constant. As a consequence, the algorithm may switch from I/O bound to CPU bound. This switch, however, does not only depend on the number m of simultaneous queries but also on the question, how early the algorithm is able to exclude pages from processing.

3.1 Problem Description

Whenever the distance of the *k*-nn candidate for some query q_i ($1 \le i \le m$) decreases, all pages P_j having a MINDIST larger than the new *k*-nn candidate distance are excluded from being processed for this query (they are pruned off APL_i). If a page is pruned for many queries but a few, the effort of the loading operation must be performed whereas

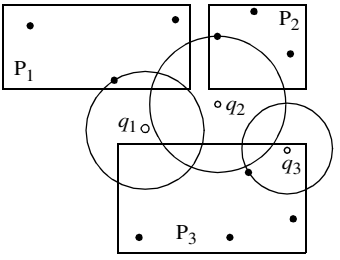


Fig. 2. Example

many processing operations are avoided in advance. Therefore, pruning saves valuable processing time and the algorithm switches back to I/O bound.

The central question we have to address is, what is a suitable order of pages that prunes as many processing operations process(P_j, q_i) as possible? For showing that this is not a trivial task, let us consider the following example in Fig. 2. Applying the HS algorithm for 1-nn queries, query q_1 would first access page P_3 and then P_1. Query q_2 would access the page sequence P_2, P_1, and P_3. Finally, q_3 would only access P_3 and then stop immediately. In multiple query processing, all three pages must be loaded into main memory, because each page is relevant for at least one query. Considering the sequence, however, makes a big difference with respect to the processing operations. For instance, a very bad choice is to start with P_2. After loading P_2, we have to perform all distance calculations between the points on P_2 and all query points since using the triangle inequality cannot avoid any distance calculation. Additionally, the data points found on page P_2 do not exclude any of the remaining pages P_1 and P_3 from any query, because the current 1-nn candidates still have a high distance from the query points. Therefore, the *pruning distances* are bad, or in other words, P_2 does not yield a high *pruning power*. A much better choice is P_3 since once P_3 is loaded and processed, P_1 and P_2 are pruned for query q_3, because the pruning distance of q_3 becomes fairly low. If P_1 is loaded next, P_2 can additionally be pruned for q_1. Finally, P_2 is loaded, but only the distances to q_2 must be determined since P_2 is pruned for all other queries. Thus, we have saved 3 out of 9 processing operations compared to a scheduling sequence starting with page P_2.

3.2 The Pruning Power Definition

This simple example already demonstrates that a good page schedule is essential in order to further improve the processing of multiple k-nn queries. Thus, our general objective is to load pages which have a high *pruning power*. We define the pruning power as follows:

Definition 1: Pruning Power

The pruning power of a page P_r is the number of processing operations process(P_s, q_i), $1 \le s \le num_pages$, $s \ne r$, $1 \le i \le m$, which can be avoided if page P_r is loaded and processed.

According to this definition, the pruning power of a page is exactly known only at the time when the page has been processed. Recall that processing operations can be avoided only if the k-nn candidate (and, thus, the pruning distance) for some query changes.

In general, it is not possible to determine in advance whether such a change occurs. Therefore, our goal is first to capture more precisely the conditions under which a page has a high pruning power, and then to develop heuristics which select pages that obey these conditions. The core problem is to determine an expected value r_{prune} for the radius for which the intersection volume contains a number k' of points:

$$r_{\text{prune}} = \int_0^\infty \left(r \cdot \frac{\partial}{\partial r} \left(\sum_{x=k'}^{C_{\text{eff}}} \binom{C_{\text{eff}}}{x} \cdot \left(\frac{V_{\text{inters}}(r)}{V_{\text{pagereg}}} \right)^x \cdot \left(1 - \frac{V_{\text{inters}}(r)}{V_{\text{pagereg}}} \right)^{C_{\text{eff}}-x} \right) \right) dr, \qquad (1)$$

where k' is the number of points which are needed to have a set of k points in total. A difficult task here is to determine the intersection volume, which can be solved by a tabulated Montecarlo integration [7]. Although tabulation makes this numerical method possible, the performance is not high enough, since eq. (1) must be evaluated often.

A promising approach to address such problems is to introduce heuristics which inherently follow the optimal solution. When considering single k-nn algorithms proposed in the literature, e.g. [12], [15], we can extract two substantial measures which are associated with the pruning power: the minimum distance MINDIST and the priority of a page in the corresponding APL. Both measures have been successfully used to schedule pages for a single k-nn query. Additionally, it has been shown that other measures, e.g. the minimum of the maximum possible distances from a query point to a face of a considered page region, perform only poorly. Thus, we develop three heuristics in the following section which select pages with a high pruning power on the basis of these two measures, i.e., using the priority of pages in APLs and using the distance MINDIST.

4. Pruning Power Scheduling Strategies

4.1 Average Priority Technique

This heuristic is based on the observation that a page can only have a high pruning power if it has a high local priority in several APLs. If the page has a low priority in most of the APLs, it is obviously very unlikely that this page yields a high pruning power. When selecting the next page to process, for each page P_j the ranking numbers RN(APL$_i$, P_j) in all queues APL$_i$ are determined and summed up. If, for instance, P_1 is at the first position for q_1, at the fifth position for q_2 and at the 13th position for q_3, we get a cumulated ranking number of 19 for P_1. This number is divided by the number (3 in this example) of APLs that contain P_1 in order to form an average and to avoid the underestimation of pages that are already pruned for many queries. The page with the lowest average ranking $\overline{P}$ is selected to be loaded and processed next. With npq_i we denote the number of priority queues which contain page P_i.

$$\overline{P} = \text{MIN}\left(\frac{1}{npq_1} \cdot \sum_{i=1}^{m} \text{RN}(\text{APL}_i, P_1), \ldots, \frac{1}{npq_s} \cdot \sum_{i=1}^{m} \text{RN}(\text{APL}_i, P_s) \right) \qquad (2)$$

4.2 Average Distance Technique

The average distance strategy is similar to the average priority heuristic. The focus of this approach is not the position of a page in the APLs, but directly the minimum dis-

tance $MD(q_i, P_j)$ between a page region P_j and a query point q_i. The motivation is the observation that the pruning power of a page is monotonically decreasing with increasing distance to the query point, i.e. when a page is located far away from most query points, this page probably has a low pruning power. For each page, the distances to all query points (except those for which the considered page is already pruned) are determined and summed up. Again, the average is formed by dividing the cumulated distance by the number of all queries that did not yet prune the page. The page with the least average distance $\overline{P}$ is selected for being processed next.

$$\bar{P} = \text{MIN}\left(\frac{1}{npq_1} \cdot \sum_{i=1}^{m} \text{MD}(q_i, P_1), \ldots, \frac{1}{npq_s} \cdot \sum_{i=1}^{m} \text{MD}(q_i, P_s)\right) \quad (3)$$

4.3 Maximum Priority Technique

The motivation for this strategy is the observation that pages with maximum priority (i.e. they are at the top position) in one or more priority queues often have a high pruning power even if they have very low priority for other queries. The reason is that pages with maximum priority generally are more likely to contain one of the k nearest neighbors than the pages on the following positions. Therefore, a page which is at the first position for few queries may outperform a page that is at the second position for many queries. Like the average priority technique, the maximum priority technique determines the ranking of all pages with respect to the position in the priority queues. In contrast to the average priority technique, it counts the number $\text{MPC}(\text{APL}_1, \ldots, \text{APL}_m, P_j)$ of queries q_i, for which the page P_j has the maximum priority. The page yielding the highest count $\overline{P}$ is selected as the next page to be processed. When two or more pages have an equal count, we consider position-two (and subsequent) counts as a secondary criterion.

$$\bar{P} = \text{MAX}\ (\text{MPC}(\text{APL}_1, \ldots, P_1), \ldots, \text{MPC}(\text{APL}_1, \ldots, P_s)) \quad (4)$$

5. Experimental Evaluation

In order to determine the most efficient scheduling technique we performed an extensive experimental evaluation using the following databases:

- *Synthetic database*: 1,600,000 8-*d* random points following a uniform distribution.
- *CAD database*: 16-*d* Fourier points corresponding to contours of 1,280,000 industrial parts used in the S3-system [4].
- *Astronomy database*: 20-*d* feature vectors of 1,000,000 stars and galaxies which are part of the so-called Tycho catalogue [11].

All experiments presented in this section were performed on an Intel Pentium II-300 workstation under Linux 6.0. The index structure we used was a variant of the X-tree where the directory consists of one large supernode. We used a block size of 32 KBytes for the X-tree and the cache size was set to 10% of the X-tree size.

For index structures with a hierarchically organized directory, the proposed scheduling techniques can be applied as dynamic approaches or as hybrid approaches (sequences of static schedules). However, with the directory consisting of one large supernode, we are able to apply the techniques in a purely static way: Since we can completely construct the APL once a query point is provided, we can also determine a page schedule

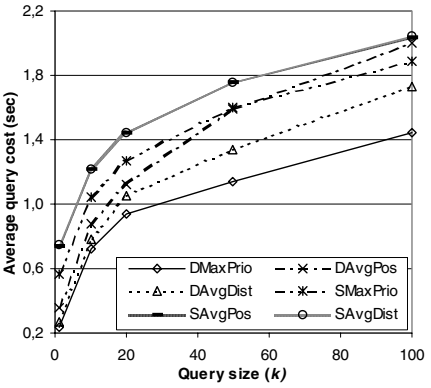


Fig. 3. Query size on the CAD database

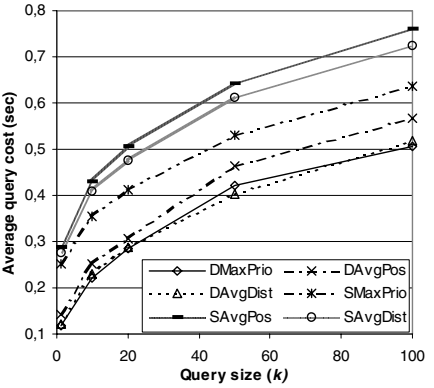


Fig. 5. Query size on the synthetic database

before we start query processing. While this is straightforward for the average priority and the average distance technique, we have to slightly modify the maximum priority technique. First, we determine all pages having a maximum priority and sort them with respect to their counts. Assuming that those pages are already processed, we consider all pages located on the second position of any APL as pages with maximum priority and again sort them. This step iterates until all pages are enlisted in the schedule. Additionally, for all static approaches we check if a chosen page is still needed by any query before loading the page. We experimentally evaluate the following page scheduling techniques (cf. section 4):

- Static and Dynamic Average Priority (denoted as *SAvgPos* and *DAvgPos*)
- Static and Dynamic Average Distance (denoted as *SAvgDist* and *DAvgDist*)
- Static and Dynamic Maximum Priority (denoted as *SMaxPrio* and *DMaxPrio*)

We first investigate the effect of the query size using the CAD database. The maximum number m of multiple k-nn queries in the system is set to 20 and we make the simplifying assumption that at any time there are enough queries in a queue waiting to be processed to load the system completely. The query size k varies and Fig. 3 depicts the average query cost including the CPU and I/O cost. We can observe that *DMaxPrio* clearly outperforms all other scheduling techniques for all values of k. Compared to the second best technique *DAvgDist*, the *DMaxPrio* approach exhibits 85% of the average query cost for $k = 1$ and 83% of the average query cost for $k = 100$. Considering the *SAvgPos* and the *SAvgDist* approaches (their performance plots are almost identical), the average query cost of *DMaxPrio* is only 32% (70%) of the corresponding average query cost for 1- (100-) nearest neighbor queries. All dynamic techniques outperform the static approaches up to $k = 50$. For $k > 50$, *SMaxPrio* starts to outperform the *DAvgPos* approach. With increasing k, the performance gain of *DMaxPrio* compared to the other techniques decreases. The reason is that with increasing query size, the distance to the k-nn (pruning distance) increases and fewer distance calculations can be avoided by applying the triangle inequality. However, even for high values of k the *DMaxPrio* approach saves 12% - 30% of the query cost compared to the other techniques.

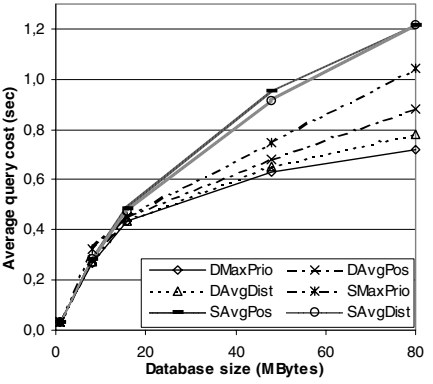


Fig. 6. Database size on the CAD database

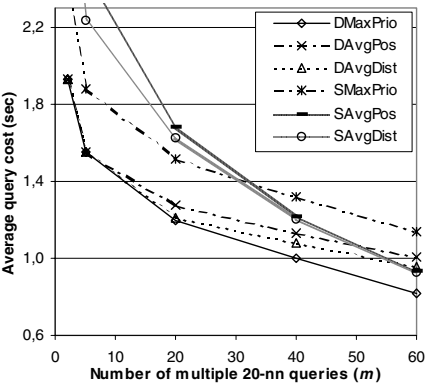


Fig. 7. # of mult. queries (Astronomy db)

The same experiment is performed on the synthetic database (cf. Fig. 5). As before, the *DMaxPrio* approach yields the best overall performance and leads to an average query cost of 36% - 59% for $k = 1$ and 7% - 34% for $k = 100$ of the average query cost of the other scheduling techniques except for *DAvgDist*. In this experiment, the *DAvgDist* approach shows a comparative performance as *DMaxPrio* and even outperforms *DMaxPrio* for $k = 50$. For all values of k, the dynamic techniques provide a much better performance than the static techniques. Considering only the static approaches, we observe that *SMaxPrio* outperforms *SAvgPos* and *SAvgDist*.

Next, we analyze the impact of the database size on the scheduling techniques. We used the CAD database and increased the number of Fourier points from 12,800 up to 1,280,000. We kept the maximum number m of multiple k-nn queries at 20 and performed 10-nn queries (cf. Fig. 6). For small database sizes, all scheduling techniques show similar performance. When increasing the database size, this situation changes drastically: For database sizes larger than 16 MBytes, the performance of *SAvgPos* and *SAvgDist* degenerates whereas *DMaxPrio* and *DAvgDist* show a comparatively moderate increase of the average query cost (*DMaxPrio* again outperforms *DAvgDist*). This can be explained by the following observation: With increasing database size, the average length of the APLs also increases and the static approaches more and more suffer from the lack of information resulting from processing candidate pages. The average query cost of *SMaxPrio* shows an acceptable increase up to 48 MBytes. For 80 MBytes, however, also this approach exhibits poor performance. Considering the dynamic scheduling techniques, *DAvgPos* has the worst performance for large database sizes.

Since dynamic approaches generally introduce some computational overhead for determining the page schedule, we also investigated the system parameter m which typically depends on hardware aspects and on the underlying application. We used the astronomy database and performed 20-nn queries while increasing the maximum number m of multiple queries (cf. Fig. 7). For all scheduling techniques, the average query cost clearly decreases with increasing m which underlines the effectiveness of the multiple query approach. Again, *DMaxPrio* yields the best overall performance (it outperforms all other techniques for $m \geq 5$). An important result is the following observation: While

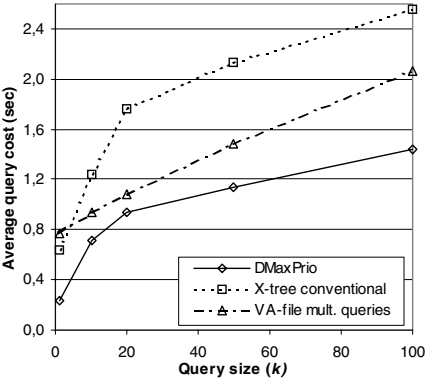


Fig. 8. Effectiveness of the multiple query approach on the CAD database

for small values of m the dynamic approaches obviously outperform the static approaches, we observe that *SAvgPos* and *SAvgDist* outperform *DAvgPos* and *DAvgDist* for $m = 60$. The reason for this result is the increasing cost of dynamically calculating the scheduling criterion since the number of APLs that must be analyzed is directly proportional to the number of queries in the system. The *DMaxPrio* approach, on the other hand, exhibits excellent performance even for high values of m due to the fact that the decision criterion can be evaluated at almost no extra cost, since in average only the top elements of each APL have to be analyzed.

The objective of our last experiment is to show the efficiency of our approach in general. We compared the most efficient pruning power scheduling technique (*DMaxPrio*) with the multiple queries technique using an efficient variant of the linear scan, namely the VA-file [17], and we compared it with conventional query processing using the X-tree. For this experiment, we used the CAD database, set m to 20 and varied the query parameter k from 1 to 100. The average query cost with respect to k is depicted in Fig. 8. Our new scheduling technique clearly outperforms the conventional X-tree query processing by speed-up factors ranging from 2.72 for 1-nn queries to 1.78 for 100-nn queries. Considering the VA-file using the multiple query scheme (*VA-file mult. queries*), we can observe that the average query cost of *DMaxPrio* is less than the average query cost of the multiple query VA-file for all values of k. However, this does not hold for all scheduling techniques. For instance, using *DAvgPos* or a static approach (e.g. *SMaxPrio*) for the page scheduling, the multiple query VA-file outperforms the multiple query X-tree already for $k > 10$ (compare with Fig. 3). This result underlines the importance of finding an efficient and robust scheduling technique in order to maximize the performance improvement resulting from the multiple query scheme.

6. Conclusions

In this paper, we have studied the problem of page scheduling for multiple k-nn query processing prevalent in data mining applications such as proximity analysis, outlier identification or nearest neighbor classification. We have derived the theoretic foundation and found that the pruning power of a page is the key information in order to solve the scheduling

problem. The pruning power results from the distance between the *k*-nn candidates located in a data page and the query points. We have proposed several scheduling algorithms which base on the pruning power theory. An extensive experimental evaluation demonstrates the practical impact of our technique. For future work, we plan to analyze our technique in parallel and distributed environments.

References

1. Berchtold S., Böhm C., Keim D., Kriegel H.-P.: *'A Cost Model for Nearest Neighbor Search in High-Dimensional Data Spaces',* Proc. 16th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, Tucson, USA, 1997, pp. 78-86.
2. Braunmüller B., Ester M., Kriegel H.-P., Sander J.: '*Efficiently Supporting Multiple Similarity Queries for Mining in Metric Databases*', Proc. 16th Int. Conf. on Data Engineering, San Diego, USA, 2000, pp. 256-267.
3. Berchtold S., Böhm C., Jagadish H.V., Kriegel H.-P., Sander J.: '*Independent Quantization: An Index Compression Technique for High-Dimensional Spaces*', Proc. Int. Conf. on Data Engineering, San Diego, USA, 2000, pp. 577-588.
4. Berchtold S., Kriegel H.-P.: *'S3: Similarity Search in CAD Database Systems',* Proc. ACM SIGMOD Int. Conf. on Management of Data, Tucson, USA, 1997, pp. 564-567.
5. Berchtold S., Keim D., Kriegel H.-P.: *'The X-tree: An Index Structure for High-Dimensional Data',* Proc. Conf. on Very Large Data Bases, Mumbai, India, 1996, pp. 28-39.
6. Breunig M. M., Kriegel H.-P., Ng R., Sander J.: 'OPTICS-OF: Identifying Local Outliers', Proc. Conf. on Principles of Data Mining and Knowledge Discovery, Prague, 1999, in: Lecture Notes in Computer Science, Springer, Vol. 1704, 1999, pp. 262-270.
7. Böhm C.: *'Efficiently Indexing High-Dimensional Data Spaces',* Ph.D. thesis, University of Munich, Munich, Germany, 1998.
8. Friedman J. H., Bentley J. L., Finkel R. A.: '*An Algorithm for Finding Best Matches in Logarithmic Expected Time*', ACM Transactions on Mathematical Software, Vol. 3, No. 3, 1977, pp. 209-226.
9. Fayyad U. M., Piatetsky-Shapiro G., Smyth P.: '*From Data Mining to Knowledge Discovery: An Overview*', Advances in Knowledge Discovery and Data Mining, AAAI Press, 1996, pp. 1 - 34.
10. Gaede V., Günther O.:'*Multidimensional Access Methods'*, ACM Computing Surveys, Vol. 30, No. 2, 1998, pp.170-231.
11. Høg E. et al.: "*The Tycho Catalogue*", Journal of Astronomy and Astrophysics, Vol. 323, 1997, pp. L57-L60.
12. Hjaltason G. R., Samet H.: '*Ranking in Spatial Databases*', Proc. Int. Symp. on Large Spatial Databases, Portland, USA, 1995, pp. 83-95.
13. Knorr E.M., Ng R.T.: '*Finding Aggregate Proximity Relationships and Commonalities in Spatial Data Mining*,' IEEE Transactions on Knowledge and Data Engineering, Vol. 8, No. 6, pp. 884-897.
14. Mitchell T. M.: '*Machine Learning*', McGraw-Hill, 1997.
15. Roussopoulos N., Kelley S., Vincent F.: '*Nearest Neighbor Queries*', Proc. ACM SIGMOD Int. Conf. on Management of Data, San Jose, USA, 1995, pp. 71-79.
16. Samet H.: '*The Design and Analysis of Spatial Data Structures*', Addison-Wesley, 1989.
17. Weber R., Schek H.-J., Blott S.: '*A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces*', Proc. Int. Conf. on Very Large Data Bases, New York, USA, 1998, pp. 194-205.

Data Mining Support in Database Management Systems

Tadeusz Morzy, Marek Wojciechowski, Maciej Zakrzewicz

Poznan University of Technology, Poland
{morzy,marek,mzakrz}@cs.put.poznan.pl

Abstract. The most popular data mining techniques consist in searching databases for frequently occurring patterns, e.g. association rules, sequential patterns. We argue that in contrast to today's loosely-coupled tools, data mining should be regarded as advanced database querying and supported by Database Management Systems (DBMSs). In this paper we describe our research prototype system, which logically extends DBMS functionality, offering extensive support for pattern discovery, storage and management. We focus on the system architecture and novel SQL-based data mining query language, which serves as the user interface to the system.

1 Introduction

The primary goal of data mining is to discover frequently occurring, previously unknown, and interesting patterns from large databases [8]. The discovered patterns are usually represented in the form of association rules or sequential patterns. The results of data mining are mostly used to support decisions, observe trends, and plan marketing strategies. For example, the association rule "A & F -> D" states, that the purchase of the products A and F is often associated with the purchase of the product D.

From the conceptual point of view, data mining can be perceived as advanced database querying, since the resulting information in fact exists in the database, but it is difficult to retrieve it. For this reason, there is a very promising idea of integrating data mining methods with DBMSs [14][17], where users specify their problems by means of data mining queries. This leads to the concept of on-line data mining, fully supported by the DBMS architecture (similarly to OLAP).

We have built a research prototype system, called *RD2*, which logically extends DBMS functionality and allows users to mine relational databases. Users or user applications communicate with *RD2* by means of *MineSQL* language, used to express data mining problems. *MineSQL* is a multipurpose, declarative language, based on *SQL*, for discovering, storing and managing association rules and sequential patterns. The novel ideas of data mining views and materialized views are also implemented in *RD2* and supported by *MineSQL*. In this paper we focus on *MineSQL* language and its usage in the area of data mining.

Y. Kambayashi, M. Mohania, and A M. Tjoa (Eds.): DaWaK 2000, LNCS 1874, pp. 382–392, 2000.
© Springer-Verlag Berlin Heidelberg 2000

1.1 Basic Definitions

Association rules
Let $L=\{l_1, l_2, ..., l_m\}$ be a set of literals, called items. Let a non-empty set of items T be called an *itemset*. Let D be a set of variable length itemsets, where each itemset $T \subseteq L$. We say that an itemset T *supports* an item $x \in L$ if x is in T. We say that an itemset T *supports* an itemset $X \subseteq L$ if T supports every item in the set X.

An *association rule* is an implication of the form $X \rightarrow Y$, where $X \subset L$, $Y \subset L$, $X \cap Y = \emptyset$. Each rule has associated measures of its statistical significance and strength, called *support* and *confidence*. The rule $X \rightarrow Y$ holds in the set D with support s if $100*s\%$ of itemsets in D support $X \cup Y$. The rule $X \rightarrow Y$ has confidence c if $100*c\%$ of itemsets in D that support X also support Y.

Sequential patterns
Let $L = \{l_1, l_2, ..., l_m\}$ be a set of literals called items. An *itemset* is a non-empty set of items. A *sequence* is an ordered list of itemsets and is denoted as $<X_1 X_2 ... X_n>$, where X_i is an itemset ($X_i \subseteq L$). X_i is called an *element* of the sequence. Let D be a set of variable length sequences, where for each sequence $S = <X_1 X_2 ... X_n>$, a timestamp is associated with each X_i.

With no time constraints we say that a sequence $X = <X_1 X_2 ... X_n>$ is *contained* in a sequence $Y = <Y_1 Y_2 ... Y_m>$ if there exist integers $i_1 < i_2 < ... < i_n$ such that $X_1 \subseteq Y_{i1}$, $X_2 \subseteq Y_{i2}$, ..., $X_n \subseteq Y_{in}$. We call $<Y_{i1}, Y_{i2}, ..., Y_{in}>$ an *occurrence* of X in Y. We consider the following user-specified time constraints while looking for occurrences of a given sequence: minimal and maximal gap allowed between consecutive elements of an occurrence of the sequence (called *min-gap* and *max-gap*), maximal duration (called *time window*) of the occurrence and time *tolerance*, which represents the maximal time distance between two itemsets to treat them as a single one.

A *sequential pattern* is a sequence whose statistical significance in D is above user-specified threshold. We consider two alternative measures of statistical significance for sequential patterns: *support* and *number of occurrences*. The support for the sequential pattern $<X_1 X_2 ... X_n>$ in the set of sequences D is the percentage of sequences in D that contain the sequential pattern. While counting the support it is not important how many times a pattern occurs in a given data sequence. This makes support unsuitable when sequential patterns are mined over a single data sequence ($|D| = 1$). In such case, the number of occurrences is more useful as a statistical measure.

Generalized patterns
In many applications, interesting patterns between items often occur at a relatively high concept level. For example, besides discovering that "40% of customers who purchase *soda_03*, also purchase *potato_chips_12*", it could be informative to also show that "100% of customers who purchase *any of beverages* also purchase *potato_chips_12*". Such association rules or sequential patterns utilize conceptual hierarchy information and are called, respectively, *generalized association rules* and *generalized sequential patterns*.

A *conceptual hierarchy*, also called *taxonomy*, consists of a set of nodes organized in a tree, where nodes in the tree represent values of an attribute, called *concepts*. The

generalized value of an attribute is the replacement of its value with its ancestor located on a given *level* in the conceptual hierarchy. Conceptual hierarchies are provided by domain experts or users, or generated automatically.

1.2 Related Work

The problem of mining association rules was first introduced in [1] and an algorithm called *AIS* was proposed. In [3], two new algorithms were presented, called *Apriori* and *AprioriTid* that are fundamentally different from previous ones. In [6] a new algorithm, called Max-Miner, was introduced to efficiently mine long association patterns. In [5] an algorithm exploiting all user-specified statistical constraints (including minimum support and confidence) was presented. In [10] and [15] the problem of finding generalized (also called multiple-level) association rules based on a user-defined taxonomy was addressed.

The problem of mining frequent patterns in a set of data sequences together with a few mining algorithms was first introduced in [4]. The class of patterns considered there, called sequential patterns, had a form of sequences of sets of items. The statistical significance of a pattern (called support) was measured as a percentage of data sequences containing the pattern. In [16], the problem was generalized by adding taxonomy on items and time constraints such as min-gap, max-gap and sliding window (in this paper called tolerance).

Another formulation of the problem of mining frequently occurring patterns in sequential data was given in [13], where discovered patterns (called episodes) could have different type of ordering: full (serial episodes), none (parallel episodes) or partial and had to appear within a user-defined time window. The episodes were mined over a single event sequence and their statistical significance was measured as a percentage of windows containing the episode (frequency) or as a number of occurrences.

In [11], the issue of integrating data mining with current database management systems was addressed and a concept of KDD queries was introduced. Several extensions of *SQL* were presented to handle association rules queries [7][9][12]. In [12], storing of discovered rules in a rulebase was discussed.

In recent years many data mining projects have been developed. Some of them resulted in commercial products. The two particularly interesting data mining systems are *DBMiner* [9] and *IBM Intelligent Miner* (which evolved from the *Quest* project [2]). They are both multi-purpose data mining tools that can be used to discover various frequently occurring patterns in data but they also offer support for high-level tasks such as classification or clustering. Both tools have a graphical user interface that can be used to select the source dataset, the desired data mining method and parameters required by the selected method. In addition, *DBMiner* offers a *SQL*-like data mining query language (*DMQL*) that can be used to specify data mining tasks. Although the systems currently available offer some level of integration with DBMSs, they do not provide mechanisms for storage and management of data mining results.

2 RD2 Architecture

RD2 is a data mining system for relational databases, built initially on top of Oracle database management system. *RD2* discovers association rules and sequential patterns in database tables, according to users' needs expressed in the form of data mining queries. The system uses *RD2* Network Adapter to communicate with user applications and *RD2* Database Adapter to communicate with the database management system. The *RD2* system can work in 2-tier architecture, residing on the database computer, as well as in 3-tier architecture, when residing on a separate computer. Figure 1 gives an overview of *RD2* system architecture.

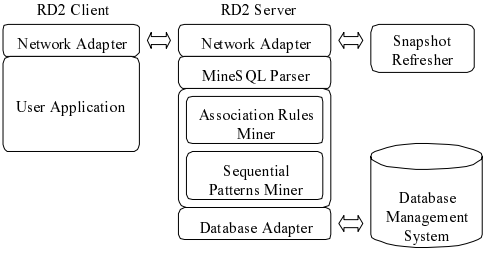


Fig. 1. RD2 Architecture

Fig. 2. RD2 FrontEnd application

RD2 Network Adapter is a module for network communication between a client application and *RD2* server. It uses TCP/IP protocol to transmit data mining queries generated by the client application to the server and to transmit the discovered association rules or sequential patterns from the server back to the client application. *RD2* Network Adapter contains the programmer's interface (*RD2 API*), which is used by client applications, cooperating with *RD2*. Advanced users can also use a tool, called *RD2* FrontEnd, which is an *RD2* application for ad-hoc data mining. Users can execute their data mining queries and watch the results on the screen (see Figure 2).

RD2 Database Adapter provides transparent access to various database management systems. Primarily implemented on Oracle DBMS (using Oracle Call Interface), *RD2* architecture is independent on the DBMS vendor. *RD2* Database Adapter translates its API calls into native DBMS functions. The adapter can communicate with both local and remote DBMSs.

The client application requests are expressed in the form of *MineSQL* data mining queries. *RD2* **MineSQL Parser** is used for syntactic and semantic analysis of the queries. It builds the parse tree for a query and then calls the appropriate query processing procedures.

Association Rules Miner is a module for discovering all association rules which satisfy user-specified predicates. The association rules are discovered from the database accessed via Database Adapter while the predicates are extracted from *MineSQL* queries by *MineSQL* Parser. Our own constraints-driven algorithm for association rules discovery is used to perform the task.

Sequential Patterns Miner is a module for discovering all sequential patterns which satisfy user-specified predicates. The sequential patterns are discovered from the database accessed via Database Adapter while the predicates are extracted from *MineSQL* queries by *MineSQL* Parser.

RD2 Snapshot Refresher is an independent module responsible for updating materialized views defined on data mining queries. It re-executes the view queries in regular time intervals and stores the refreshed view contents in the database.

3 MineSQL Data Mining Query Language

MineSQL is a declarative language for expressing data mining problems. It is a *SQL*-based interface between client applications and *RD2* system. *MineSQL* plays similar role to data mining applications as *SQL* does to database applications. *MineSQL* is *declarative* - the client application is separated from the data mining algorithm being used. Any modifications and improvements done to the algorithm do not influence the applications. *MineSQL* follows the syntax philosophy of *SQL* language – data mining queries can be combined with *SQL* queries, i.e. *SQL* results can be mined and *MineSQL* results can be queried. Thus, existing database applications can be easily modified to use *RD2* data mining. *MineSQL* is also *multipurpose* – it can be used for different types of frequent patterns: association rules, sequential patterns, generalized association rules, and generalized sequential patterns. *MineSQL* also supports storage of the discovered patterns in the database – both association rules and sequential patterns can be stored in database tables similarly to alphanumeric data. Repetitive data mining tasks can be optimized by means of data mining views and materialized views.

3.1 New SQL Data Types

MineSQL language defines a set of new *SQL* data types, which are used to store and manage association rules, sequential patterns, itemsets and itemset sequences. The new data types can be used for table column definitions or as program variable types.

The *SET OF* data types family is used to represent sets of items, e.g. a shopping cart contents. *SET OF NUMBER* data type represents a set of numeric items, *SET OF CHAR* represents a set of character items, etc. Below we give an example of a statement, which creates a new table to store basket contents.

```
CREATE TABLE SHOPPING (ID NUMBER, BASKET SET OF CHAR);
```

In order to convert single item values into a *SET OF* value, we use a new *SQL* group function called *SET*. Below we give an example of a *SQL* query, which returns the *SET OF* values from normalized market baskets.

```
SELECT SET(ITEM)
FROM PURCHASED_ITEMS GROUP BY T_ID;
```

SET(ITEM)
A
A,C,D

PURCHASED_ITEMS:

T_ID	ITEM
1	A
2	A
2	C
2	D

We define the following *SQL* functions and operators for the *SET OF* data types: *ITEM(x,i)* - the *i*-th item value of the set *x* (lexicographical ordering presumed), *SIZE(x)* - the number of items in the set *x*, *s CONTAINS q* - *TRUE* if the set *s* contains the set *q*, *s UNION q* - the union of the sets *s* and *q*, *s MINUS q* - the difference of the sets *s* and *q*, *s INTERSECT q* - the intersection of the sets *s* and *q*.

The *RULE OF* data types family is used to represent association rules, containing body, head, support and confidence values. *RULE OF NUMBER* data type represents association rules on numeric values (e.g. "1 & 3 -> 5"), *RULE OF CHAR* data type represents association rules on character values (e.g. "beer & diapers -> chips"), etc. The following statement creates a new table to store association rules.

```
CREATE TABLE MYRULES (ID NUMBER, AR RULE OF CHAR);
```

A user can insert an association rule into the table manually, using a new *SQL* function called *TO_RULE*, e.g.:

```
INSERT INTO MYRULES (ID,AR) VALUES(15, TO_RULE('A,D','B,C',0.8,0.1));
SELECT * FROM MYRULES;
```

ID	AR
15	A & D -> B & C (0.8;0.1)

We also define a set of the following *SQL* functions and operators that operate on rules: *BODY(x)* - the *SET OF* value representing the body of the rule *x*, *HEAD(x)* - the *SET OF* value representing the head of the rule *x*, *SUPPORT(x)* - support of the rule *x*, *CONFIDENCE(x)* - confidence of the rule *x*, *s [NOT] SATISFIES x* - *TRUE* if the set *s* satisfies the rule *x*, *s [NOT] VIOLATES x* - *TRUE* if the set *s* violates the rule *x*.

The following example query displays all sets from *PURCHASED_ITEMS* table, which violate the association rule "A & D -> B & C & G".

```
SELECT SET(ITEM)
FROM PURCHASED_ITEMS GROUP BY T_ID
HAVING SET(ITEM) VIOLATES TO_RULE('A,D','B,C,G',0.8,0.1);
```

The *SEQUENCE OF* data types family is used to represent sequences of sets of items, e.g. histories of customers' purchases. Sequences are ordered collections of (timestamp, value) pairs, where timestamp is usually of date and time type and value can be a set of elements of any type. For example, *SEQUENCE OF CHAR INDEX BY DATE* data type represents a sequence of sets of character items ordered according to timestamps of date type. Below we give an example of a statement, which creates a new table to store purchase histories.

```
CREATE TABLE SHOPPING_HIST
(CUST_ID NUMBER, HIST SEQUENCE OF CHAR INDEX BY DATE);
```

In order to convert a collection of (timestamp, value) pairs into a *SEQUENCE OF* value, we use a new *SQL* group function called *SEQUENCE*. The example below shows a query returning the *SEQUENCE OF* values from normalized market baskets.

```
SELECT SEQUENCE(T_TIME, ITEM)
FROM CUST_TRANSACTIONS GROUP BY C_ID;
```

SEQUENCE(T_TIME, ITEM)
<Feb 20,2000;(A)> <Feb 21,2000;(D)>
<Feb 20,2000;(B,D)> <Feb 22,2000;(A)>

CUST_TRANSACTIONS:

T_TIME	C_ID	ITEM
Feb 20, 2000	10	A
Feb 20, 2000	20	B
Feb 20, 2000	20	D
Feb 21, 2000	10	D
Feb 22, 2000	20	A

A user can insert a sequence into the table manually, using a new *SQL* function called *TO_SEQUENCE*. The following example presents the appropriate *INSERT* statement and the resulting contents of *SHOPPING_HIST*.

```
INSERT INTO SHOPPING_HIST (CUST_ID, HIST)
VALUES (51, TO_SEQUENCE('<Feb 20,2000;(A)> <Feb 21,2000;(D,F)>'));
```

We also define a set of the following *SQL* functions and operators that operate on sequences: *ELEMENT(x,i)* - the *i*-th element value of the sequence *x*, *LENGTH(x)* - the number of elements in the sequence *x*, *SIZE(x)* - the number of items in the sequence *x*, *s CONTAINS t* - *TRUE* if the sequence *s* contains the sequence *t*, *s CONTAINS t MAXGAP x* - *TRUE* if the sequence *s* contains the sequence or pattern *t* and the interval between adjacent matching elements in *s* is not greater than x, *s CONTAINS t MINGAP x* - *TRUE* if the sequence *s* contains the sequence or pattern *t* and the interval between adjacent matching elements in *s* is not less than x, *s CONTAINS t WINDOW x* - *TRUE* if the sequence *s* contains the sequence or pattern *t* within the time window of *x*, *s CONTAINS t TOLERANCE x* - *TRUE* if the sequence *s* contains the sequence or pattern *t* with the tolerance of *x*.

The *PATTERN OF* data types family is used to represent sequential patterns and their statistical significance (support or number of occurrences). *PATTERN OF CHAR* data type represents patterns on character values (e.g. "<(TV) (VCR) (DVD)>"), *PATTERN OF NUMBER* data type represents patterns on numeric values (e.g. "<(10 20 30) (40 50)>"), etc. Below we give an example of a statement, which creates a new table to store sequential patterns.

```
CREATE TABLE MYPATTERNS(ID NUMBER,SP PATTERN OF NUMBER);
```

A user can insert a sequential pattern into the table manually, using the new *SQL* function called *TO_PATTERN*. The following example presents the appropriate *INSERT* statement and the resulting contents of *MYPATTERNS*.

```
INSERT INTO MYPATTERNS (ID, SP)
VALUES (15, TO_PATTERN('<(10 20) (30)>',0.3, null));
```

We also define a set of the following *SQL* functions and operators that operate on sequential patterns: *ELEMENT(x,i)* - the *i*-th element of the pattern *x*, *LENGTH(x)* - the number of elements in the pattern *x*, *SIZE(x)* - the number of items in the pattern *x*, *SUPPORT(x)* - support of the pattern *x*, *OCCURRENCES(x)* - number of occurrences of the pattern *x*, *p CONTAINS s* - *TRUE* if the pattern *p* contains the sequence *s*.

The following query displays all patterns from MYPATTERNS, which contain the subsequence <(10)(30)> (notice the dynamic sequence creation in the example).

```
SELECT SP FROM MYPATTERNS
WHERE SP CONTAINS TO_SEQUENCE('<1;(10)><2;(30)>');
```

3.2 Mining Association Rules

The central statement of the *MineSQL* language is *MINE*. *MINE* is used to discover association rules or sequential patterns from the database. *MINE* also specifies a set of predicates to be satisfied by the returned rules or patterns. In order to discover association rules we use the following syntax of *MINE* statement.

```
MINE rule_expression [, rule_expression…] FOR column [, column …]
FROM {table|(query)} WHERE rule_predicate [AND rule_predicate…];
```

where: *rule_expression* is the keyword *RULE* or a function operating on *RULE* (*RULE* represents a single association rule being discovered), *column* is the name of the table column or query column of the type *SET OF*, containing itemsets to be mined; when specifying multiple columns, then the itemsets are combined (columns must be of the same data type), *table* is the name of a table containing the itemsets to be mined, *query* is the *SQL* subquery, returning the itemsets to be mined, *rule_predicate* is a Boolean predicate on a function which operates on *RULE,* to be satisfied by returned association rules.

The following *MINE* statement uses the *PURCHASED_ITEMS* table to discover all association rules, whose support is greater than 0.1 and confidence is greater that 0.3. We display the whole association rules, their bodies and supports.

```
MINE RULE, BODY(RULE), SUPPORT(RULE)
FOR X FROM (SELECT SET(ITEM) AS X
            FROM PURCHASED_ITEMS GROUP BY T_ID)
WHERE SUPPORT(RULE)>0.1 AND CONFIDENCE(RULE)>0.3;
```

3.3 Mining Sequential Patterns

In order to discover sequential patterns we use the following syntax of *MINE* statement.

```
MINE patt_expression [, patt_expression…]
[WINDOW window][MAXGAP maxgap][MINGAP mingap][TOLERANCE tolerance]
FOR column FROM {table|(query)}
WHERE patt_predicate [AND patt_predicate…];
```

where: *patt_expression* – the keyword *PATTERN* or a function operating on *PATTERN* (*PATTERN* represents a single sequential pattern being discovered), *window* is the time window size, *maxgap* is the maximal gap allowed between consecutive elements of an occurrence of the sequence, *mingap* is the minimal gap allowed between consecutive elements of an occurrence of the sequence, *tolerance* is the time tolerance value for pattern elements, *column* is the name of the table column or query column of the type *SEQUENCE OF*, containing sequences to be mined, *table* is the name of a table containing the sequences to be mined, *query* is the *SQL* subquery, returning the sequences to be mined, *patt_predicate* is a Boolean predicate on a function which operates on *PATTERN*, to be satisfied by returned sequential patterns.

The following *MINE* statement uses the *CUST_TRANSACTIONS* table to discover all sequential patterns, whose support is greater than 0.1. We display the patterns and their supports.

```
MINE PATTERN, SUPPORT(PATTERN)
FOR X FROM (SELECT SEQUENCE(T_TIME, ITEM) AS X
            FROM CUST_TRANSACTIONS GROUP BY C_ID)
WHERE SUPPORT(PATTERN)>0.1;
```

3.4 Using Views and Materialized Views

Relational databases provide users with a possibility of creating views and materialized views. A view is a virtual table presenting the results of the *SQL* query hidden in the definition of the view. Views are used mainly to simplify access to frequently used data sets that are results of complex queries. When a users selects data from a view, its defining query has to be executed but the user does not have to be familiar with its syntax.

Since data mining tasks are repetitive in nature and the syntax of data mining queries may be complicated, we propose to extend the usage of views to handle both *SQL* queries and *MineSQL* queries. The following statement creates the view presenting the results of one of the data mining tasks discussed earlier.

```
CREATE VIEW BASKET_RULES
AS MINE RULE, BODY(RULE), SUPPORT(RULE)
FOR X FROM (SELECT SET(ITEM) AS X
            FROM PURCHASED_ITEMS GROUP BY T_ID)
WHERE SUPPORT(RULE)>0.1;
```

Any *SQL* query concerning the view presented above involves performing the data mining task according to the data mining query that defines the view. This guarantees access to up-to-date patterns but leads to long response times, since data mining algorithms are time consuming. In database systems it is possible to create materialized views that materialize the results of the defining query to shorten response times. Of course, data presented by a materialized view may become invalid as the source data changes. One of the solutions minimizing effects of this problem is periodic refreshing of materialized views.

We introduce materialized data mining views with the option of automatic periodic refreshing. A materialized data mining view is a database object containing patterns (association rules or sequential patterns) discovered as a result of a data mining query. It contains rules and patterns that were valid at a certain point of time. Materialized data mining views can be used for further selective analysis of discovered patterns with no need to re-run mining algorithms. Using materialized views is easier than creating a table with columns of type *RULE OF* or *PATTERN OF* and filling it with results of a data mining query. Moreover, materialized views offer additional functionality, because they can be automatically refreshed according to a user-defined time interval. This might be useful when a user is interested in a set of rules or sequential patterns, whose specification does not change in time, but always wants to have access to relatively recent information.

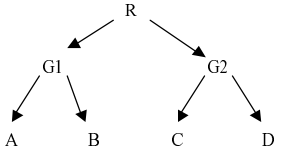
To create a materialized data mining view we use the following syntax.

```
CREATE MATERIALIZED VIEW view_name
[REFRESH time_interval] AS mine_statement
```

In the above syntax *view_name* is the name of a materialized view, *time_interval* denotes the time interval between two consecutive refreshes of the view (in days), and *mine_statement* denotes any variation of the *MINE* statement. The *REFRESH* clause is optional since a user might not want a view to be refreshed automatically.

3.5 Taxonomies and Generalized Patterns

In order to discover generalized association rules and generalized sequential patterns, *MineSQL* allows users to define conceptual hierarchies. A conceptual hierarchy, or a taxonomy, is a persistent database object created by means of a *CREATE TAXONOMY* and *INSERT* statements. The following example illustrates a conceptual hierarchy (called *MY_TAX*) and the statements for its definition.




```
CREATE TAXONOMY MY_TAX OF CHAR;
INSERT INTO MY_TAX NODE 'R';
INSERT INTO MY_TAX NODE 'G1' REFERENCES 'R';
INSERT INTO MY_TAX NODE 'G2' REFERENCES 'R';
INSERT INTO MY_TAX NODE 'A' REFERENCES 'G1';
INSERT INTO MY_TAX NODE 'B' REFERENCES 'G1';
INSERT INTO MY_TAX NODE 'C' REFERENCES 'G2';
INSERT INTO MY_TAX NODE 'D' REFERENCES 'G2';
```

After a conceptual hierarchy has been created, *MINE* statements can be extended to use it. Additional keyword *USING* is used to specify the name of a conceptual hierarchy to be used for the associated attribute. The following example data mining query discovers all generalized association rules between values of the attribute *ITEM* in the database table *PURCHASED_ITEMS*, grouped by the *T_ID* attribute. Values of the attribute *ITEMS* are generalized by means of the conceptual hierarchy called *MY_TAX*.

```
MINE RULE FOR ITEMS USING MY_TAX
FROM (SELECT SET(ITEM) AS ITEMS
      FROM PURCHASED_ITEMS GROUP BY T_ID)
WHERE SUPPORT(RULE)>0.3
```

Conceptual hierarchies also influence previously presented set, rule, sequence, and sequential pattern operators: *s CONTAINS q USING c - TRUE* if the set *s* contains the set *q* according to the conceptual hierarchy *c*, *s [NOT] SATISFIES x USING c - TRUE* if the set *s* satisfies the rule *x* according to the conceptual hierarchy *c*, *s [NOT] VIOLATES x USING c - TRUE* if the set *s* violates the rule *x* according to the conceptual hierarchy *c*, *s CONTAINS t USING c - TRUE* if the sequence *s* contains the sequence *t* according to the conceptual hierarchy *c*, *p CONTAINS s USING c - TRUE* if the pattern *p* contains the sequence *s* according to the conceptual hierarchy *c*.

4 Concluding Remarks

In the paper we have presented our research prototype system, which logically extends DBMS functionality to mine relational databases. We introduced a data mining query language, called *MineSQL*, which supports pattern discovery, storage and management in relational environment. A number of illustrative examples for various *MineSQL* statements have been presented.

In the future we plan to extend the results of our research along the following lines: 1. DBMS support for data mining in object-oriented databases, focused on using inheritance hierarchies as taxonomies and mining of polymorphic collections, 2. Re-

search on data mining query optimization methods, focused on materialized views group refreshing and physical data structures, 3. Study whether other data mining methods should be supported by *MineSQL*, e.g. data classification, clustering.

Bibliography

1. Agrawal R., Imielinski T., Swami A.: Mining Association Rules Between Sets of Items in Large Databases. Proc. of the ACM SIGMOD Conference on Management of Data (1993)
2. Agrawal R., Mehta M., Shafer J., Srikant R., Arning A., Bollinger T.: The Quest Data Mining System. Proc. of the 2nd KDD Conference (1996)
3. Agrawal R., Srikant R.: Fast Algorithms for Mining Association Rules. Proc. of the 20th Int'l Conf. on Very Large Data Bases (1994)
4. Agrawal R., Srikant R.: Mining Sequential Patterns. Proc. of the 11th Int'l Conference on Data Engineering (1995)
5. Bayardo R. J., Agrawal R., Gunopulos D.: Constraint-Based Rule Mining in Large, Dense Databases. Proc. of the 15th Int'l Conference on Data Engineering (1999)
6. Bayardo R. J.: Efficiently Mining Long Patterns from Databases. Proc. of the ACM SIGMOD International Conference on Management of Data (1998)
7. Ceri S., Meo R., Psaila G.: A New SQL-like Operator for Mining Association Rules. Proc. of the 22nd Int'l Conference on Very Large Data Bases (1996)
8. Fayyad U., Piatetsky-Shapiro G., Smyth P.: The KDD Process for Extracting Useful Knowledge from Volumes of Data. Communications of the ACM, Vol. 39, No. 11 (1996)
9. Han J., Fu Y., Wang W., Chiang J., Gong W., Koperski K., Li D., Lu Y., Rajan A., Stefanovic N., Xia B., Zaiane O.R.: DBMiner: A System for Mining Knowledge in Large Relational Databases. Proc. of the 2nd KDD Conference (1996)
10. Han J., Fu Y.: Discovery of Multiple-Level Association Rules from Large Databases. Proc. of the 21st Int'l Conf. on Very Large Data Bases (1995)
11. Imielinski T., Mannila H.: A Database Perspective on Knowledge Discovery. Communications of the ACM, Vol. 39, No. 11 (1996)
12. Imielinski T., Virmani A., Abdulghani A.: Datamine: Application programming interface and query language for data mining. Proc. of the 2nd KDD Conference (1996)
13. Mannila H., Toivonen H., Verkamo A.I.: Discovering frequent episodes in sequences. Proc. of the 1st KDD Conference (1995)
14. Morzy T., Zakrzewicz M.: SQL-like Language for Database Mining. Proc. of the 1st ADBIS Conference (1997)
15. Srikant R., Agrawal R.: Mining Generalized Association Rules. Proc. of the 21st Int'l Conf. on Very Large Data Bases (1995)
16. Srikant R., Agrawal R.: Mining Sequential Patterns: Generalizations and Performance Improvements. Proc. of the 5th Int'l Conf. on Extending Database Technology (1996)
17. Wojciechowski M.: Mining Various Patterns in Sequential Data in an SQL-like Manner. Proc. of the 3rd ADBIS Conference (1999)

Decision trees for probabilistic data

Jean-Pascal ABOA*, Richard EMILION†

Abstract

We propose an algorithm to build decision trees when the observed data are probability distributions. This is of interest when one deals with massive database or with probabilistic models. We illustrate our method with a dataset describing districts of Great Britain. Our decision tree yields rules which explain the unemployment rate.
The decision tree in our case is built by replacing the test $X > \alpha$, which is used to split the nodes in the usual case of real numbers, by the test $P(X > \alpha) < \beta$, where α and β are determined through an algorithm based on probabilistic split evaluation criteria.

1 Introduction

Decision trees are particularly suited to data mining (see e.g [1], [6]). Generally, they are constructed from a set of objects whose attributes are numerical or categorial deterministic variables. In the present paper we present a decision tree construction method for objects described by probabilistic data. Such descriptions naturally appear because of the phenomenal growth of datasets size. This argument is illustrated in the example in section 2.

More generally, probability distribution functions (p.d.f.) appear when the initial data are inaccurate or if one deals with probabilistic models. We thus see that it is a very natural (and also theoretically interesting) problem to extend standard statistic or data analysis methods which usually work for real data to the case of more complex data such as p.d.f.. This is emphasized for example in [5], [8].

The rest of this paper is organized as follows. Section 2 presents an example of G.B. districts described by probabilistic data. In section 3, we introduce our basic setting. Section 4 is devoted to the decision tree construction algorithm. This yields probabilistic rules which explain the unemployment rate of each district. A conclusion is given in section 5.

*Ceremade, Univ. Paris IX-Dauphine, 75775 Paris cedex 16, aboa@ceremade.dauphine.fr
†UFR SEGMI modalx, univ. Paris X, Richard.Emilion@modalx.u-paris10.fr

Y. Kambayashi, M. Mohania, and A M. Tjoa (Eds.): DaWaK 2000, LNCS 1874, pp. 393–398, 2000.
© Springer-Verlag Berlin Heidelberg 2000

2 An example : G.B. districts

We start with a table of N rows and p columns. All the entries f_{ij} on a fixed column j are probabilites on a common space V_j which only depends on j. Let $N = 135$ districts of Great Britain (G.B.) and $p = 6$ descriptors :

X_1 denotes the *age* descriptor (in years) :
$X_1 = 1$ for $0 - 4$, $X_1 = 2$ for $5 - 14$, $X_1 = 3$ for $15 - 24$, $X_1 = 4$ for $25 - 44$, $X_1 = 5$ for $45 - 64$, $X_1 = 6$ for greater than or equal to 65.

X_2 denotes the *racial origin* :
$X_2 = 1$ for Whites, $X_2 = 2$ for Blacks, $X_2 = 3$ for Asians, $X_2 = 4$ otherwise.

X_3 denotes the *number of children* :
$X_3 = 1$ for less than 4 children, $X_3 = 2$ for more than 4 children.

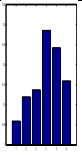
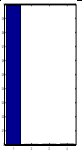
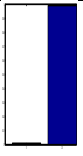
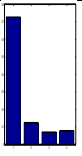
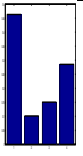
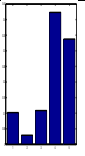
X_4 denotes the *accomodation type* :
$X_4 = 1$ for owner occupied accomodation, $X_4 = 2$ for public sector accomodation, $X_4 = 3$ for private sector accomodation, $X_4 = 4$ in other cases.

X_5 denotes the *social class of household head* :
$X_5 = 1$ if head is in class 1 or 2 , $X_5 = 2$ if head is in class 3, $X_5 = 3$ if head is in class 4 or 5, $X_5 = 4$ for the others.

X_6 denotes the *occupation domain* :
$X_6 = 1$ for Agriculture, $X_6 = 2$ for Primary Production, $X_6 = 3$ for manufacturing, $X_6 = 4$ for Service, $X_6 = 5$ in other cases.

Districts	Age	Origin	Number of kids
Hambleton	6% 12% 14% 28% 24% 16%	99.6% 0.1% 0.1% 0.2%	1.0% 99.0%

Districts	Accomodation	Social class	Occupation
Hambleton	73.0% 12.0% 7.0% 8.0%	46% 10% 15% 29%	10% 3% 11% 42% 34%

	Age	Origin	Nber of kids	Accomodat.	Social c.	Occupation
Hambleton						

The *classifying* attribute Y is the *unemployment rate for men and women.* In our application this rate fluctuates between 0% and 18%. We define two prior classes : class 1 for rate $\leq 9\%$ and class 0 for rate $> 9\%$.

3 Basic setting

In a general situation we will consider a table $(f_{ij})_{i=1,\ldots N, j=1,\ldots p}$ of probabilities on sets V_j as described in section 2. Let o_i denote the object number i described by the i^{th}-row $f_i = (f_{i1}, \ldots f_{ip})$. The whole set $\{o_1, \ldots, o_N\}$ will be denoted by O. In example of section 2, o_i is nothing but the i^{th}-district.

It will be convenient to suppose that in each column j, f_{ij} is the probability distribution of a random variable $X_{ij} : (\Omega, P) \to V_j$.
The row random vectors $X_{i.} = (X_{i1}, \ldots, X_{ip})$ are independant (but not the column vectors). For each j let $\mathcal{V}_j$ be a finite class of subsets of V_j which will be used to split the decision tree nodes.

In the example of section 2, we can take $\mathcal{V}_j$ as the power set of V_j. But if V_j is large we select relevant subsets by generalizing the result in ([3] p. 274).
The classifying variable Y is a categorial attribute. We let $Y(o_i) = y_i$.

4 The method and the algorithm

The root (starting node) of the decision tree is the whole set O. Given a node, we split it into two children in the best way possible. The splitting is determined by an attribute X_j and the best j is choosen according to a splitting quality criterion. We repeat the procedure on each child.
So decision trees successively divide the set of training until all the subsets consisting of data belong, entirely or predominantly, to a single class.

4.1 Dichotomy

Let X_j be an attribute, and let $I \in \mathcal{V}_j$ and $\beta \in (0, 1)$. The dichotomy induced by (X_j, I, β) is defined by :

$$D_{j,I,\beta} = \{\mathcal{N}_1, \mathcal{N}_2\} \tag{1}$$

$$\mathcal{N}_1 = \{o_i : P(X_{ij} \in I) > \beta\} \quad \text{and } \mathcal{N}_2 = \{o_i : P(X_{ij} \in I) \leq \beta\}$$

4.2 Goodness of a split

Our method uses the entropy index [7] to derive the splitting criterion at every internal node of the tree. To each node $\mathcal{N}$ is associated a *homogeneity index* $i_{\mathcal{N}}$ defined by :

$$i_{\mathcal{N}} = -p_{0/\mathcal{N}} log_2 p_{0/\mathcal{N}} - p_{1/\mathcal{N}} log_2 p_{1/\mathcal{N}} \tag{2}$$

where $p_{0/\mathcal{N}}$ (resp. $p_{1/\mathcal{N}}$) is the proportion of objects in $\mathcal{N}$ belonging to class 0 (resp. class 1).

Each division $D_{j,I,\beta} = \{\mathcal{N}_1, \mathcal{N}_2\}$ of the node $\mathcal{N}$, induces a homogeneity increase defined by the following number :

$$\Delta\left(D_{j,I,\beta}\right) = i_{\mathcal{N}} - \{P_{\mathcal{N}_1} i_{\mathcal{N}_1} + P_{\mathcal{N}_2} i_{\mathcal{N}_2}\} \tag{3}$$

where $i_{\mathcal{N}}$, $i_{\mathcal{N}_1}$, $i_{\mathcal{N}_2}$ are the homogeneity index of nodes $\mathcal{N}$, $\mathcal{N}_1$, $\mathcal{N}_2$ respectively, and $P_{\mathcal{N}_1}$, $P_{\mathcal{N}_2}$ are the proportion of objects assigned to $\mathcal{N}_1$, $\mathcal{N}_2$ respectively after partitioning.

For each node $\mathcal{N}$ and each j, we select $I_j \in \mathcal{V}_j$ and $\beta_j \in (0,1)$ for which $\Delta\left(D_{j,I_j,\beta_j}\right)$ is maximal. We finally select the "best" X_{j^*} among the p attributes by :

$$\Delta\left(D_{j^*,I_{j^*},\beta_{j^*}}\right) = \max_{j,I_j,\beta_j} \Delta\left(D_{j,I,\beta}\right) \tag{4}$$

4.3 Algorithm

Let $\varepsilon > 0$ be a fixed integer
Starting node $\mathcal{N} = \{o_1, \ldots, o_N\}$
Partition(node $\mathcal{N}$)
    If $((100 - \varepsilon)\%$ of objects in $\mathcal{N}$ are of the same class) then
        return;
    For each j
        begin
            For each I_{jr}, $r = 1, \ldots, n_j$
                begin
                determine the best β_{jr} according to the test
                $P\left(X_{.j} \in I_j r\right) > \beta_{jr}$
                End
            determine the "best" (I_j, β_j) among (I_{jr}, β_{jr}), $r = 1, \ldots, n_j$
        End
    determine the "best" (I_{j^*}, β_{j^*}) among (I_j, β_j), $j = 1, \ldots p$, which split node $\mathcal{N}$ into two nodes $\mathcal{N}_1$ and $\mathcal{N}_2$.

Partition(node $\mathcal{N}_1$)
Partition(node $\mathcal{N}_2$)

4.4 Application

The above algorithm yields the following tree and rules.

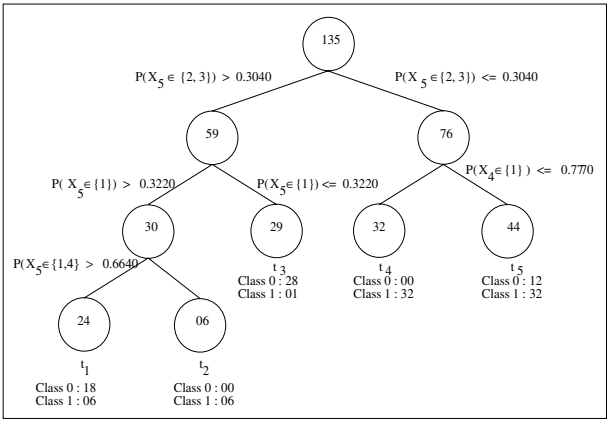


Figure 1: Decision tree for the G.B. districts

Rule 1 : if % (residents in households where head is class 3, 4 or 5) $> 30\%$ and % (residents in households where head is class 1 or 2) $> 32\%$ and % (residents in households where head is class 1 or 2 or other cases) $> 66\%$ then the unemployment rate is high with 75% chance and weak with 25% chance
Rule 2 : if % (residents in households where head is class 3, 4 or 5) $> 30\%$ and % (residents in households where head is class 1 or 2) $> 32\%$ and % (residents in households where head is class 1 or 2 or other cases) $\leq 66\%$ then the unemployment rate is weak with 100% chance
Rule 3 : if % (residents in households where head is class 3, 4 or 5) $> 30\%$ and % (residents in households where head is class 1 or 2) $\leq 32\%$ then the unemployment rate is high with 97% chance and weak with 3% chance
Rule 4 : if % (residents in households where head is class 3, 4 or 5) $\leq 30\%$ and % (residents in owner occupied accomodation) $> 78\%$ then the unemployment rate is weak with 100% chance
Rule 5 : if % (residents in households where head is class 3, 4 or 5) $\leq 30\%$ and % (residents in owner occupied accomodation) $\leq 78\%$ then the unemployment rate is high with 27% chance and weak with 73% chance

Table 1: Probabilistic rules

Remark : Chavent [4] has proposed a similar algorithm with $\beta = 0.5$ but $\beta = 0.5$ need not be the most pertinent threshold. Indeed the test $P(X_5 \in \{2,3\}) > 0.3040$ gives an index of 0.2989, while for $\beta = 0.5$ the "best" test $P(X_5 \in \{3,4\}) > 0.5$ gives an index of only 0.2989.

5 Conclusion

We have presented a decision tree construction method for probabilistic data. This seems of interest when we deal with very large dataset. Our algorithm has been implemented in C language with a dataset of G.B. districts.
Our main new technical point concerns the selection of subsets in the splitting test. This may lead to further research.

References

[1] K. Alsabti, S. Ranka, and V.Singh. Clouds: A decision tree classifier for large datasets. In *KDD'98*, Août 1998.

[2] A. Baccini and A. Pousse. Point de vue unitaire de la segmentation. quelques conséquences. *CRAS*, A(280):241, Janvier 1975.

[3] L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone. *Classification and Regression trees*. Wadsworth and brooks, 1984.

[4] M. Chavent. *Analyse des données symboliques: une méthode divisive de classification*. PhD thesis, Université Paris 9 Dauphine, 1998.

[5] E. Diday. Introduction à l'approche symbolique en analyse des données. *Cahier du CEREMADE, univ. Paris Dauphine, N. 8823*, 1988.

[6] R. Kohavi and M. Sahami. Error-based and entropy-based discretization of continuous features. In *KDD'96*, 1996.

[7] J.R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1), 1986.

[8] Schweizer. Distribution functions : numbers of future. In *Mathematics of fuzzy systems*, pages 137–149. 2nd Napoli meeting, 1985.

Mining Frequent Binary Expressions

Toon Calders* and Jan Paredaens

Universiteit Antwerpen,
Departement Wiskunde-Informatica,
Universiteitsplein 1, B-2610 Wilrijk, Belgium.
{calders,pareda}@uia.ua.ac.be

Abstract. In data mining, searching for frequent patterns is a common basic operation. It forms the basis of many interesting decision support processes. In this paper we present a new type of patterns, *binary expressions*. Based on the properties of a specified binary test, such as reflexivity, transitivity and symmetry, we construct a generic algorithm that mines all frequent binary expressions.
We present three applications of this new type of expressions: mining for rules, for horizontal decompositions, and in intensional database relations.
Since the number of binary expressions can become exponentially large, we use data mining techniques to avoid exponential execution times. We present results of the algorithm that show an exponential gain in time due to a well chosen pruning technique.

1 Introduction

In data mining, searching for frequent patterns is a basic operation. It forms the basis of many interesting decision support processes. Most data mining algorithms first start searching frequent patterns. In association rule mining [1], frequent itemsets are mined.

In this paper we will present a new type of patterns, *binary expressions*, that will be the basis of three applications. A binary expression is a conjunction of binary tests between attributes. An example of such an expression using the test $<$ is $(1 < 2) \wedge (2 < 3)$, expressing that attribute 1 is smaller than attribute 2 and attribute 2 is smaller than attribute 3. A binary expression will be called *frequent* iff the number of tuples satisfying the expression is bigger than a given threshold. Based on the properties of a specified binary test, such as reflexivity, transitivity and symmetry, we construct a generic algorithm that searches all frequent binary expressions. The properties are used to avoid syntactically different, but semantically equal expressions. The following two different expressions

$$(1 < 2) \wedge (2 < 3)$$

$$(1 < 2) \wedge (2 < 3) \wedge (1 < 3)$$

* Research Assistant of the Fund for Scientific Research - Flanders (Belgium)(F.W.O. - Vlaanderen).

Y. Kambayashi, M. Mohania, and A M. Tjoa (Eds.): DaWaK 2000, LNCS 1874, pp. 399–408, 2000.
© Springer-Verlag Berlin Heidelberg 2000

will select exactly the same tuples. This is due to the fact that the binary test $<$ is transitive. We will give a method to avoid generating both expressions.

In this paper we present three applications that have the mining of frequent binary expressions in common. The first one is rule mining. A *binary association rule* is a rule $X \rightarrow Y$, where X and Y are binary expressions. Just like in association rule mining, we define notions of *support* and *confidence* for this type of rules. The similarities with association rules will be elaborated in Section 3.

The second application is in making horizontal decompositions. Horizontal decompositions have already been studied extensively[3] and are important in the context of distributed databases. When we want to make a horizontal decomposition, it is important to find a good criterion to split the relation. We will use the frequent binary expressions to make an optimal decomposition of a relation, based on target sizes of the fragments.

A third application is mining with *intensional database relations*[4]. In *Inductive Logic Programming* (ILP), the mining base typically contains intensional relations besides the traditional extensional relations. In this context, in the mining process, the intensional relations can be viewed as tests, in addition to the traditional tests such as $<, =, \ldots$, and rules that contain intensional relations can be mined in much the same way as other tests.

The outline of the paper will be as follows: in Section 2 binary expression, equivalence of expressions and some other notions are formally defined. In Section 3 the applications mentioned above are studied. In Section 4 we give some properties of the search space of the algorithm. In Section 5 we present a generic algorithm to find all frequent binary expressions. In Section 6 some experimental result of the algorithm are given. These results show good scalability properties of the algorithm. Section 7 concludes the paper.

An extended version of this paper is available as [2].

2 Definitions

Before we elaborate the three applications given in the introduction, we define formally the notions of respectively a relation, a binary test, a binary expression and equivalence of expressions

First we fix the relations we will consider. We only look at relations where all attributes have the same domain $\mathcal{U}$ [1]. $\mathcal{U}$ is a, possibly infinite, recursive set. We use an unnamed perspective; i.e. we will refer to the attributes by their number.

Definition 1. *An $n-ary$ relation is a finite subset of $\mathcal{U}^n$.*
A binary test[2] *θ over $\mathcal{U}$ is a recursive subset of $\mathcal{U} \times \mathcal{U}$. When $(u_1, u_2) \in \theta$, we will write $u_1 \theta u_2$.*

We now define the notion of an expression.

[1] $\mathcal{U}$ stands for *Universe*.

[2] We use the name *binary test* instead of *relation*, to avoid confusion with *database relations*. Actually, a binary test is just a relation in the mathematical sense.

Definition 2. *Let θ be a binary test. A (θ, n)-expression (θ and n will be omitted when clear from the context) is a conjunction of $(i\theta j)$'s, where $1 \leq i, j \leq n$. The set of all (θ, n)-expressions will be denoted by $\mathcal{E}(\theta, n)$.*

The previous definition gave the syntax of an expression. The next definition gives the semantics of expressions.

Definition 3. *Let $\mathcal{R}$ be an n-ary relation, e is a (θ, n)-expression. $\sigma_e\mathcal{R} = \{r \in \mathcal{R} \mid (\forall (i\theta j)$ in $e) r(i)\theta r(j)\}$[3] is the* selection on e of $\mathcal{R}$.

We are now ready to state the problem of mining all frequent binary expressions.

Definition 4. *Let $\mathcal{R}$ be an n-ary relation. The* frequency *of the binary expression $e \in \mathcal{E}(\theta, n)$, denoted $freq(e, \mathcal{R})$ is $\frac{|\sigma_e \mathcal{R}|}{|\mathcal{R}|}$.*
Let t be a number between 0 and 1. A binary expression $e \in \mathcal{E}(\theta, n)$ is t-frequent iff $freq(e, \mathcal{R}) \geq t$. ($t$ will be omitted if clear from the context.)
The solution of the $freq(\mathcal{R}, t, \theta)$-problem is the set of all t-frequent (θ, n)-expressions.

Example 1. Consider the following relation:

1	2	3	4
1	2	1	4
1	5	6	2
1	5	1	1

The solution of the $freq(\mathcal{R}, \frac{2}{3}, <)$-problem is the set $\{1 < 2, 1 < 4, 3 < 2, 4 < 2, 1 < 2 \wedge 1 < 4, 1 < 2 \wedge 3 < 2, 1 < 2 \wedge 4 < 2\}$.

3 Applications

In this section we describe three applications of mining frequent binary expressions.

3.1 Rule Discovery

First we define a binary association rule.

Definition 5. *A* binary association rule *is a rule $X \rightarrow Y$, where X and Y are (θ, n)-expressions.*
The support *of the rule $X \rightarrow Y$ is $freq(X \wedge Y, \mathcal{R})$.*
The confidence *of the rule $X \rightarrow Y$ is $\frac{freq(X \wedge Y, \mathcal{R})}{freq(X, \mathcal{R})}$.*

Example 2. Consider the relation given in Example 1. The support of the binary association rule $1 < 4 \rightarrow 1 < 2$ is $\frac{2}{3}$. The confidence is 1.

[3] $r(i)$ denotes the i-th component of r; e.g. $(a, b, c)(2) = b$.

There are multiple similarities between association rules and binary association rules. Both rules give frequent dependencies that hold within the tuples themselves. Unlike for example roll-up dependencies [9], that describe relations between different tuples, association rules and binary association rules relate properties of attributes. In association rule mining, frequent itemsets can be considered as a conjunction of unary predicates. In this setting, binary association rules are a straightforward extension of the unary predicates to binary predicates. A binary association rule finds associations between binary predicates, where association rules find associations between unary predicates.

3.2 Horizontal Decompositions

Horizontal decompositions are very important for distributed databases. In many cases it is desirable to fragment the database over different locations. In that case it is important to find good criteria to divide the database. We will call this a split-problem. The solution to a split-problem is an expression that selects a fraction of the tuples whose cardinality is as close to the given goal as possible.

Example 3. Consider the relation given in Example 1. $3 < 2$ is a solution for the split-problem where the goal is $\frac{1}{2}$, and the binary test $<$, since $|\sigma_{3<2}\mathcal{R}|$ is as close to $\frac{|\mathcal{R}|}{2}$ as possible.

3.3 Intensional Database Relations

In *Inductive Logic Programming* (ILP) [4], mining conjunctions with intensional relations besides extensional relations is very common. The mining base used in Logic Programming typically contains a number of extensional relations and some intensional relations. The intensional relations are given by a set of describing rules in a logic programming language, for example Prolog or Datalog. In the context of mining, the intensional relations can be viewed as tests, in addition to the traditional tests such as $<, =, \ldots$

Example 4. Suppose the following logic program is given:

```
Related(X,Y):-Parent(X,Y);
Related(X,Z):-Related(X,Y) & Related(Y,Z);
Related(X,Y):-Related(Y,X);
Related(X,X);
```

From the last three rules we can conclude that the binary relation `Related` is transitive, symmetric, and reflexive.

In this example the intensional relation `Related` is in fact a binary test. Using this similarity, we can apply all results we obtain for mining binary expressions to this case. Suppose for example than we have a predicate `King` and we use the binary test `Related`. We could for example find that the expression `Related`(X, Y)&`King`(X)&`King`(Y) is frequent. Because we know that `Related` is symmetric, we know that testing `Related`(X, Y)&`Related`(Y, X)&`King`(X)& `King`(Y) is redundant.

4 The Search Space

The *freq*$(\mathcal{R}, t, \theta)$-problem is essentially a search-problem. We want to find all frequent binary expressions in the search space $\mathcal{E}(\theta, n)$. For all binary tests θ, the number of expressions in $\mathcal{E}(\theta, n)$ is $2^{(n^2)}$, since the number of pairs of attributes is n^2, and for every pair (x, y), $x\theta y$ is present or absent. However, it is not always necessary to consider all expressions. When there are equivalent expressions, there is no need to consider them all.

Example 5. $1 = 2 \wedge 1 = 3$ is equivalent to $1 = 2 \wedge 2 = 3$.

In Tab. 1, for some binary tests and different number of attributes, the total number of non-equivalent elements in the search space is given. For example, for the equality and 3 attributes, the search space is $\{1 = 1, 1 = 2, 1 = 3, 2 = 3, 1 = 2 = 3\}$. Therefore, in Tab. 1, the row for $n = 3$ contains 5, the size of the search space. The value of $2^{(n^2)}$ is also given for each value of n.

Table 1. Size of the search space for some binary tests

n	$<$	$\leq$	$\neq$	$=$	2^{n^2}
1	2	1	2	1	2
2	3	4	2	2	16
3	19	29	8	5	512
4	219	355	64	15	65536

To exploit the equivalence of expressions we need some properties of the expressions to decide when two expressions are equivalent. Based on these properties we will construct a mechanism to avoid generation of equivalent expressions.

Definition 6. *A binary test θ has property*
P_1 = reflexive *iff for all* $1 \leq i \leq n$, $(i\theta i)$ *holds.*
Q_1 = anti-reflexive *iff for all* $1 \leq i \leq n$, $(i\theta i)$ *does* not *hold.*
P_2 = symmetric *iff for all* $1 \leq i, j \leq n$, *if* $(i\theta j)$ *then also* $(j\theta i)$ *holds.*
Q_2 = anti-symmetric *iff for all* $1 \leq i, j \leq n$, *if* $(i\theta j)$, *then* $(j\theta i)$ *does* not *hold.*
P_3 = transitive *iff for all* $1 \leq i, j, k \leq n$, *if* $(i\theta j)$ *and* $(j\theta k)$, *then also* $(i\theta k)$ *holds.*
Q_3 = anti-transitive *iff for all* $1 \leq i, j, k \leq n$, *if* $(i\theta j)$ *and* $(j\theta k)$, *then* $(i\theta k)$ *does* not *hold.*

From definition 6, it results that for each i, P_i holds or Q_i, or none. This means that there exist $3^3 = 27$ possible combinations. However, only 16 of them really exist.

Definition 7. *Let θ be a binary test, and let $P \subseteq \{P_1, P_2, P_3\}$ be the set of P-properties of θ. An expression $e \in \mathcal{E}(\theta, n)$* is closed *iff every conjunct $(i\theta j)$ that*

is necessary by the properties of P appears in e.
Let $Q \subseteq \{Q_1, Q_2, Q_3\}$ be the set of Q-properties of θ. An expression $e \in \mathcal{E}(\theta, n)$ is valid *iff all conjuncts that are forbidden by the properties of Q, do not appear in e.*

Example 6. Clearly, $e = (1 < 2) \wedge (2 < 3)$ is not closed since $1 < 3$ is necessary by the transitivity and does not appear in e. On the other hand $e' = (1 < 2) \wedge (2 < 3) \wedge (1 < 3)$ is closed. e and e' are both valid expressions. $(1 < 2) \wedge (2 < 1)$ is not valid, since the anti-symmetry forbids $(2 < 1)$ when $(1 < 2)$ is present.

Lemma 1. *Given a valid (θ, n)-expression e, there is a* unique *valid and closed expression e', that is equivalent with e. e' is obtained by augmenting e with all conjuncts that are necessary by the properties of P of θ. e' is called the* closure *of e.*

In example 6, e' is the closure of e. It is clear now that in every equivalence class of expressions there is a unique closed expression. Since we have to test only one expression of each equivalence class, for solving the $freq(\mathcal{R}, t, \theta)$-problem, it is sufficient to test each closed expression.

5 Algorithm

In this section we describe an algorithm that finds all frequent binary expressions given a binary test and a relation. Basically, the algorithm performs a levelwise search as described in [6]. The levelwise algorithm is a generate-and-test algorithm. It highly depends on a *monotonicity principle* saying, roughly speaking, that whenever e_1 is more specific than e_2, and the result of e_2 is too small then the result of e_1 is also too small. The next proposition states this monotonicity principle.

Proposition 1. *Let e_1 and e_2 be two expressions, $\mathcal{R}$ is a relation, and e_1 is more specific than e_2, then $|\sigma_{e_1}\mathcal{R}| \leq |\sigma_{e_1}\mathcal{R}|$.*

Consider the following situation: We want to solve the $freq(\mathcal{R}, \frac{1}{2}, <)$-problem, and we know that the expression $1 < 2$ is not frequent. Then, using Proposition 1, we know that $1 < 2 \wedge 1 < 3$ cannot be frequent, since $1 < 2 \wedge 1 < 3$ is more specific than $1 < 2$. So, in this situation there is no need to count the frequency of $1 < 2 \wedge 1 < 3$. We can *prune* the expression $1 < 2 \wedge 1 < 3$.

The search space of our algorithm will consist of all closed and valid expressions. In Fig. 1 a part of the search space for $freq(\mathcal{R}, 3, <)$ is showed. When we use the term *children of an expression*, we mean the expressions that are next more specific in the lattice.

Our algorithm will try to prune as much of the search space as possible. We start with the most general expression of our search space, and we iteratively test more specific expressions, without ever evaluating those expressions that cannot be frequent given the information obtained in earlier iterations. More precisely, the search space is traversed level by level, from general to specific. In each

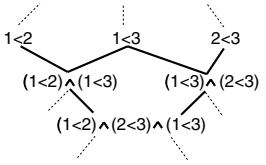


Fig. 1. A part of the search space

iteration, the set *candidates* will contain the candidate frequent expressions. An "apriori trick" is used; if the frequency of e is below the threshold, and $e' \preceq e$, then we know a priori that e' must fail the frequency threshold. For this reason, all expressions that failed the frequency threshold are stored in the set $TooLow$. This gives us the framework of Fig. 2, which actually is a *levelwise search* [6]. Steps 3 to 7 are testing the candidates against the database and bookkeeping. In step 8 the children of the frequent candidates are generated as the candidates for the next iteration. In step 9, we use the apriori trick to prune away candidates that cannot be frequent due to information obtained in previous iterations.

1. $candidates = \{\top\}$; $Output = \{\}$; $TooLow = \{\}$
2. **while**($candidates \neq \{\}$) **do**
3. **Test** Test *candidates* against the database.
4. $fcan = \{c \in candidates \mid c \text{ is frequent}\}$
5. $nfcan = candidates - fcan$
6. $Output = Output \bigcup fcan$
7. $TooLow = TooLow \bigcup nfcan$
8. **Generate** $candidates = \bigcup_{p \in fcan} \{c \mid c \text{ is a child of } p\}$
9. **Prune** $candidates = candidates - \{c \mid \exists n \in TooLow : c \preceq n\}$
10. **end while**

Fig. 2. Algorithm for finding frequent expressions

5.1 Testing

In the test-phase, the frequencies of the candidates are tested against the database. The calculation of the frequency of an expression is very costly, since we need to iterate over all tuples in the relation to count the number of tuples that satisfy the expression. To limit the overhead, all candidates in an iteration are tested in the same run over the database.

5.2 Generation

In the generation phase, we need to generate all closed and valid children of the frequent candidates. As can be seen in Fig. 1, all children are generated by

adding one conjunct, and taking the closure. However, not all conjuncts can be used for generating children; in Fig. 1, the closure of $(1 < 2)$, augmented with $(2 < 3)$ is $(1 < 2) \wedge (2 < 3) \wedge (1 < 3)$, and this is no child of $(1 < 2)$, since $(1 < 2) \wedge (1 < 3)$ lies between them. In the generation phase this problem is handled.

In the framework of the algorithm, in the generation phase, all children of the frequent candidates are generated. It is however sufficient that every expression only generates a subset of its children, as long as for every expression there is still at least one generating parent. We only generate those children that are induced by a sublattice of the search space.

Not generating all children does no harm; still all expressions are generated. On the other hand, not generating all children has a couple of advantages.

- In step 8. of the algorithm, all expressions generated by frequent candidates are added as new candidates. Probably lots of duplicates are generated. These duplicates need to be removed. The less children are generated, the less duplicates need to be removed.
- By not generating all children, some early pruning is applied. We will discuss this in more detail in the subsection on pruning.

From this discussion we can conclude that ideally each expression has exactly one generating parent. This is the case when the spanning sublattice is a tree.

The generation phase is discussed more in-depth in [2], where we introduce two strategies for the generation, ρ_1 and ρ_2.

5.3 Pruning

A basic operation of the algorithm is the pruning. It is essential that this operation is performed as efficiently as possible. The pruning implies that for every expression e that is generated in step 8 of the algorithm, we need to investigate whether there is an expression l in $TooLow$ such that $e \preceq l$. If this is the case, we can prune e.

From previous research, we can conclude that a *trie* is a good structure to store sequences. A trie uses common prefixes between the sequences to store them more efficiently. We are not going into detail on tries, for more elaborated work on tries, we refer to [5] and [7].

Step 8 is not the only step in which pruning occurs. When all *generating* parents of an expression are infrequent, the expression will never be generated, even when there are other parents that are frequent. Thus, the less parents a node has, the bigger the chance that it never will be generated if some of its parents are infrequent. This type of pruning is called *early pruning*. When an expression is not pruned early, it can still be pruned in step 8 of the algorithm. This situation occurs when at least one generating parent is frequent, and at least one other parent is infrequent. Due to the monotonicity principle the expression will be pruned.

6 Experimental Results

In this section we present some experimental results. We implemented ρ_1 and ρ_2. For a definition and a discussion of these two algorithms, we refer to [2]. The source code of both implementations can be obtained at `http://cc-www.uia.ac.be/u/calders/`.

6.1 Effectiveness of Pruning

In Fig. 3 (left), a lower bound for the total number of closed and valid expressions in $\mathcal{E}(<, n)$ is given for $n = 2, 4, \ldots, 20$ for reference. In Fig. 3 (right), some tests on a randomized dataset are given for increasing number of attributes. The number of expressions that are examined by our algorithms is given for a threshold 0.3, and for increasing number of attributes. Note that the scale of the graph representing the total size of the search space is logarithmic. The number of expressions examined by the algorithms in this example is exponentially less than the total number of elements in the search space.

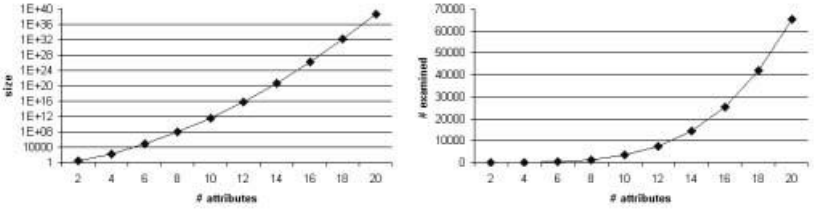


Fig. 3. The size of the search space (left) versus the number of expressions that were investigated (right)

6.2 Scalability

In Fig. 4, the running time of the two algorithms is measured. When the number of attributes grows, refinement operator ρ_2 becomes much more efficient than ρ_1. In the left graph, a threshold of 0.4 was used, and the binary test was $<$. In the right graph, the binary test $=$ was used, and the test was done with the refinement operator ρ_2, with a threshold of 0.5. In both graphs, the dataset was randomly generated. The number of values in $\mathcal{U}$ was 2 in the right graph, and 7 in the left one.

7 Conclusion

Binary expressions are an interesting type of patterns for data mining. In this paper we presented three applications of frequent binary expressions; binary

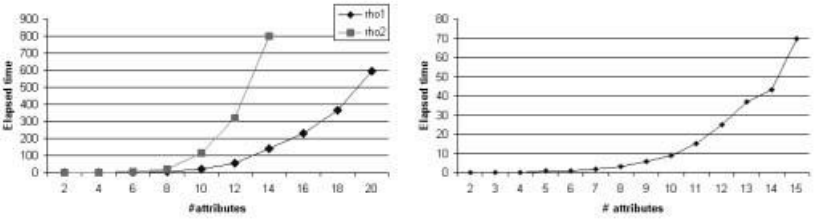


Fig. 4. Scalability in the number of attributes

rules, that essentially are extensions of association rules to binary predicates, horizontal decompositions and the mining of intensional database relations. We presented and tested an algorithm for finding frequent binary expressions. The algorithm exploited background information such as reflexivity, transitivity and symmetry of the binary tests to optimize the search.

References

[1] R. Agrawal, T. Imilienski, and A. Swami. Mining association rules between sets of items in large databases. In *Proc. ACM SIGMOD*, Washington, D.C., 1993

[2] T. Calders, and J. Paredaens. Mining Binary Expressions: Applications and Algorithms. Technical Report, Universiteit Antwerpen, Belgium, June 2000.

[3] P. De Bra. Horizontal decompositions based on functional-dependency-set-implications. In *ICDT*. Springer-Verlag, 1986.

[4] L. Dehaspe. Frequent pattern discovery in first-order logic. PhD thesis, Katholieke Universiteit Leuven, Belgium, Dec. 1998.

[5] J. Han, J.Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *Proc. ACM SIGMOD*, 2000

[6] H. Mannila and H. Toivonen. Levelwise search and borders of theories in knowledge discovery. In *Data Mining and Knowledge Discovery 1(3)*, 1997.

[7] J. Pei, J. Han, B. Mortazavi-Asl, and H. Zhu. Mining access patterns efficiently from web logs. In *PAKDD*, 2000.

[8] M. Y. Vardi. The decision problem for database dependencies. In *Inf. Proc. Letters 12(5)*, 1981.

[9] J. Wijsen, R. Ng, and T. Calders. Discovering roll-up dependencies. In *Proc. ACM SIGKDD*, 1999.

A Fast Algorithm for Hierarchical Text Classification

Wesley T. Chuang[1,3], Asok Tiyyagura[2], Jihoon Yang[3], and Giovanni Giuffrida[1]

[1] Computer Science Department, UCLA, Los Angeles, CA 90095, USA
{yelsew, giovanni}@cs.ucla.edu
[2] Department of Computer Science, Iowa State University, Ames, IA 50011, USA
asokt@cs.iastate.edu
[3] HRL Laboratories, LLC, 3011 Malibu Canyon Rd, Malibu, CA 90265, USA
{yelsew, yang}@wins.hrl.com

Abstract. Text classification is becoming more important with the proliferation of the Internet and the huge amount of data it transfers. We present an efficient algorithm for text classification using hierarchical classifiers based on a concept hierarchy. The simple TFIDF classifier is chosen to train sample data and to classify other new data. Despite its simplicity, results of experiments on Web pages and TV closed captions demonstrate high classification accuracy. Application of feature subset selection techniques improves the performance. Our algorithm is computationally efficient being bounded by $O(n\ log\ n)$ for n samples.

1 Introduction

As the amount of on-line data increases by leaps and bounds, the design of an efficient algorithm or an approach to accessing the data (e.g. through classification, clustering, filtering, etc.) has become of great interest. Two important aspects motivate such design. First, the data needs to be arranged efficiently. For example, instead of placing all the data in a flat directory, we can arrange it hierarchically based on a concept hierarchy (see Yahoo, US Patent databases, CNN and other major Internet news directories [13, 9, 2]). Querying with respect to a concept hierarchy is significantly more efficient and reliable than searching for specific keywords since the views of the data collected are refined as we go down the hierarchy [1]. Second, when text classification is our chosen approach, an efficient algorithm should be used. A number of algorithms have been proposed and their performances are compared in the literature [14].

We use the *TFIDF* text classifier [11, 4] and proceed with the following steps for hierarchical classification: The first step is to define the concept hierarchy using domain knowledge and to collect text data corresponding to the concept hierarchy. The data is then used for training the classifier and for testing the performance of our classification system. The next step is to convert the data into an appropriate form for classification (e.g. into a bag-of-words representation). Then we can derive hierarchical classifiers by supervised learning with the data collected. Finally, the classifiers are used to classify new data. Each of these steps

Y. Kambayashi, M. Mohania, and A M. Tjoa (Eds.): DaWaK 2000, LNCS 1874, pp. 409–418, 2000.
© Springer-Verlag Berlin Heidelberg 2000

is described in detail in the following sections, followed by experimental results and conclusion.

2 Concept Hierarchy

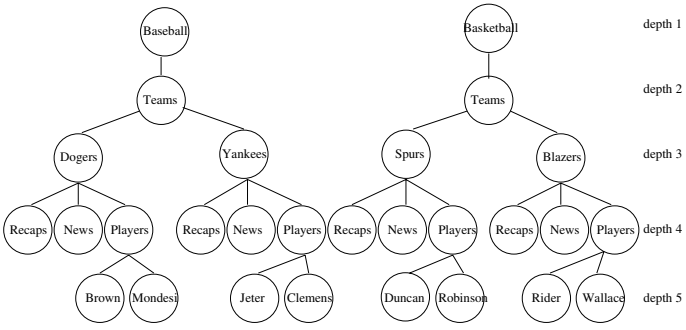


Fig. 1. A sample concept hierarchy for professional baseball and basketball.

Figure 1 shows a concept hierarchy which categorizes Web news reports for professional baseball and basketball. Initially, a concept graph is generated based on some domain knowledge. Each node in the hierarchy contains several text documents whose topic is identified as the concept.

As in relational and object-oriented databases, in which there are no absolutely standard schema (tables and classes), we do not have a faultless concept hierarchy. In fact, we should not look for one that is exclusively superior to any other even for the same data. We need one that can help us in alleviating our humans semantic burden (since a concept hierarchy encapsulates semantics) and facilitate achieving efficiency in data arrangement and searching capability.

However, unlike relational databases and object-oriented databases, Web documents, are generally unstructured. Therefore, the way we describe their schema is different from the relational and object-oriented models. Because they are unstructured, a convenient way to describe a concept in our hierarchy is to use a collection of words (features) from a document. Since it is usually very easy for humans to come up with some top-level concepts as the schema, we propose a human-generated initial concept hierarchy.

Before taking this concept hierarchy to perform feature extraction and classification, there are several assumptions to be made. First, we assume that an initial concept hierarchy has been created with each node labeled by one or a few terms representing the concept. Second, we assume that several documents have been manually placed into every node, serving as the training and testing data for *supervised learning*. Our third assumption states that a parent node owns the union of the documents of its child nodes.

3 Training the Hierarchy

After the initial hierarchy is set up and with training and testing documents placed into each node, we can proceed to "train" the hierarchy. The rationale is as follows: We will characterize those training documents residing in each node; find a range (or threshold) of the various characteristics for the node so that it can classify the test documents into certain concepts - that is, find suitable nodes to index the new documents, from top to bottom, in the concept hierarchy.

Ideally, we would hope to characterize documents in each node with a "term" or a label. However, this is not practical because for text documents, one feature term will not suffice in describing the whole document. Instead, we must "understand" the documents in some way. In the absence of a satisfactory solution to the natural language understanding problem, most current approaches to document retrieval use a *bag-of-words* representation of documents [11, 4]. It is for this reason we opt for *surface parsing* and obtain a vector (a set of word features) of weights for each document with time complexity in the order of $O(n)$, where n is the number of words in the document.

Specifically, we restrict ourselves to a relatively simple yet effective approach based on the *TFIDF* (*term frequency* $\times$ *inverse document frequency*) classifier [11, 4]. Since we are trying to separate documents into distinguishable concepts, intuitively, the combined effect of term frequency and inverse document frequency can distinguish two different document types well. Definitions of term frequency and inverse document frequency are to follow.

3.1 Representing Features for Each Node

To convert a collection of documents in each node into special representation, let's examine the documents closely. First, every document is processed using *stopping* and *stemming* procedures [11, 4] to obtain a bag of words. Stopping is the procedure to eliminate common words from the text, and stemming is the procedure to find a unique representation (e.g. root) for a word. After these procedures, we consider the following frequencies in the documents.

- *Term Frequency*: Let W be the set of words from all documents. The *term frequency* of the word w_i (ith vocabulary in W), $TF(w_i, d)$, is the number of times w_i occurs in document d.
- *Document Frequency*: $DF(w_i)$ is the number of documents in which word w_i occurs at least once.
- *Inverse Document Frequency*: $IDF(w_i) = log(\frac{|D|}{DF(w_i)})$, where $|D|$ is the total number of documents among sibling nodes.
- *Term Frequency* $\times$ *Inverse Document Frequency*: $TFIDF(w_i, d) = TF(w_i, d) \times IDF(w_i)$

We subsequently merge all child feature vectors to obtain the TF vector for a node. Calculation of TF vectors continues bottom-up until we reach the root. Meanwhile, we also propagate the DF vector for each node all the way up to the

root. When this merging process is completed, the feature vector in each node is given by: $\boldsymbol{F} = < TFIDF(w_1, d), TFIDF(w_2, d), \ldots, TFIDF(w_{|W|}, d) >$, where d is the union of documents belonging to the node, and w is the union of the set of words in d.

3.2 Determining the Threshold for Each Node

Having organized training documents into different nodes (classes) and having built feature vectors for each of them, we characterize each class by a TFIDF vector. In other words, TFIDF will now serve as a *norm* or *prototype* vector to describe that class.

Formally, let C be a collection of document classes of interest. A prototype vector $\boldsymbol{c}$ (for each class in the concept hierarchy) is generated for each class $c \in C$ as follows: $\boldsymbol{c} = \sum_{F \in c} \boldsymbol{F}$.

We will use these training documents again to make a complete hierarchical classifier in two ways. First, for each prototype vector $\boldsymbol{c}$, we need to introduce a *threshold* θ - a distance measure to indicate at what distance range to the prototype vector we consider documents fall into the same category (i.e. class). This becomes clear if we imagine that a TFIDF vector is lying in an n-dimensional hyper-space[1], shown in Figure 2. As can be seen, the threshold makes a boundary for a class. For simplicity, we visualize it after normalization (i.e. as a unit-length vector) in a 3-D coordinate system. Second, we need to examine how good the hierarchical classifier is. That is, given the prototype vectors and thresholds, what is the *accuracy* with which the classifier will correctly classify documents? Later, we will check the accuracy for other brand new test documents. However, in this section, we only consider the accuracy for the training documents which we used to come up with the prototype vectors.

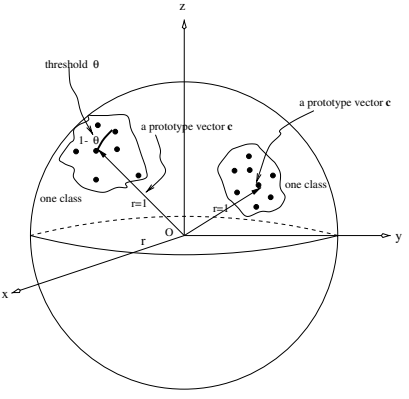


Fig. 2. Learning the threshold to admit new documents into a class.

Deriving threshold and computing training accuracy are inter-related. Here we use classification training accuracy feedback to adjust the threshold. In other words, we adjust the threshold in such a way that all documents considered for the node will yield the best accuracy. Accuracy, as shown in the following, concerns the percentage of documents correctly classifying into a class:

$$Accuracy = \frac{\#\ of\ documents\ correctly\ categorized}{\#\ of\ documents\ considered} = \frac{a+d}{a+b+c+d}$$

[1] Elements of the TFIDF vector are positive real numbers, so they can only lie in the first quadrant.

In the equation, we use a, b, c, d to indicate respectively the number of documents that should be in a class and are selected; the number selected but that should not be in; the number that should be in but are rejected; and the number that should not be in and are not selected.

Now, starting from the highest level of the hierarchy, we compare the training documents $\boldsymbol{n}$ (in vector representation) with the prototype vector of the node. When we compare two feature vectors, a *cosine* function is commonly used for similarity measure. That is, we compute the similarity of a document $\boldsymbol{n}$ to prototype vector $\boldsymbol{c}$ by: $\forall c \in C\ cos(\boldsymbol{n}, \boldsymbol{c}) = \forall c \in C\ \frac{\boldsymbol{n} \cdot \boldsymbol{c}}{\|\boldsymbol{n}\| \cdot \|\boldsymbol{c}\|}$.

```
for (level = 1 to depth of hierarchy)
    for (every node j in level)
        F_n := getPrototypeVec(Node_j);
        Docs := getDocuments(
            getSiblings(Node_j) ⋃
            getChildren(Node_j));
        < F_d > := getFeatures(Docs);
        < Dist > := cos(F_n, < F_d >);
        θ := Find a cut d in < Dist >
                that maximizes accuracy;
    end
end
```

Fig. 3. Using accuracy feedback to adjust the threshold.

For every node, we consider those training documents belonging to all sibling and child nodes. We first sort them by their distances (*Dist*) to the prototype vector. We then choose one distance that renders the best accuracy. This is the threshold, θ. Pseudocode for computing θ is shown in Figure 3. (All the functions are self-explanatory.)

3.3 Time Complexity Analysis

Our approach involves surface parsing, i.e., obtaining features for words in the documents as well as training features in the concept hierarchy. In parsing, time is linear. Most of the time is taken up by training, especially when computing the thresholds. Assume that n is the total number of documents, m is the total number of nodes in the hierarchy, and $m << n$. The contributing factors timewise are: computing TF takes $O(n)$; computing IDF takes $O(n)$; computing threshold θ takes $O(n\ log\ n)$, where $n\ log\ n$ is what the sorting costs timewise. All together, the time remains bounded by $O(n\ log\ n)$, which upholds our claim that this classification algorithm is fast.

4 Testing the Hierarchy

In this section, we test the performance of our hierarchical classifier using a test set of documents which is different from the training set. The ratio of the total number of testing documents to the training documents is about 1 to 2.

4.1 Every Node as a Classifier

When each node is equipped with a TFIDF vector and a threshold (θ), we regard the node as a *classifier*. Again, starting from the highest level of the hierarchy,

we compare the test document $\boldsymbol{n}$ (in vector representation) with the prototype vector of the node. If these two vectors are close enough, based on the cosine measure, we treat the test document as belonging to the class that the node represents, and we continue to compare the document with its child nodes. If we do not have a close match, we conclude that the document does not belong to the class and stop.

In our hierarchical classifier, a document is assigned to all the classes whose prototypes are *sufficiently* close to it. The pseudocode in Figure 4 summarizes the classification process.

Let $\boldsymbol{n}$ be the vector to be classified.
$Class = \phi$
$NodeSet = root$
while $(NodeSet \neq \phi)$ **do**
  Retrieve a $node_i$ from $NodeSet$:
    $NodeSet = NodeSet - node_i$.
  Compute the prototype vector:
    $\boldsymbol{c} = TFIDF(node_i)$.
  Compute the similarity between $\boldsymbol{n}$
    and $\boldsymbol{c}$: $s = cos(\boldsymbol{n}, \boldsymbol{c})$.
  if $(s > \theta)$
    $Class = Class \bigcup \{ c \}$.
    $NodeSet = NodeSet \bigcup$
      $\{ getChildren(node_i) \}$.
end
return $Class$

Fig. 4. Finding the class for a new text document.

4.2 Test Accuracy

Test accuracy is defined as the ratio of the number of correctly classified documents to the number of documents that are filtered at each node, i.e.,

$$test\ accuracy = \frac{\#\ correctly\ classified}{\#\ of\ filtered}$$

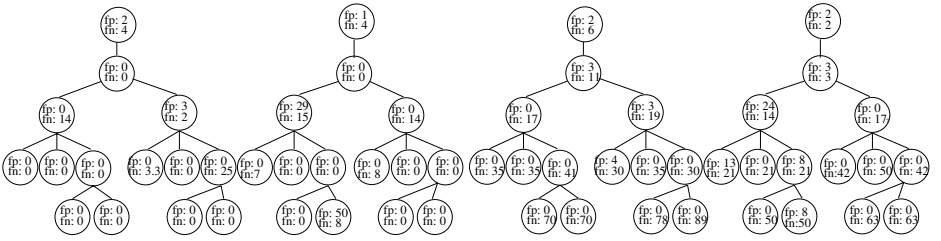
Since the classification is performed from the top to the bottom of the hierarchy, some test documents "filter" through certain nodes and continue to drill down in the hierarchy; others are stopped when their similarity measures do not pass the threshold of the prototype vectors. Consequently, the number of filtered documents diminishes as the testing process continues to the bottom level.

5 Experiments

For training and testing, we collected approximately 200 documents on professional baseball and basketball news. Experimental results demonstrate the feasibility of our approach with good classification accuracy. In training, most upper level nodes (classifiers) receive ratings of above 90% for training accuracy, while some lower level nodes perform on the average at a rate of 75%. In testing, the accuracy performance is relatively poorer, especially in lower nodes but acceptable in higher level nodes. This is because the concepts become more specific as we go down the hierarchy. In addition, the number of documents considered is relatively small compared to those at higher levels.

To account for the classification error, both in training and testing phases, we look into two different types of errors: false-positive (FP) and false-negative (FN). They are defined as follows (using a, b, c, and d as explained before):

$false\text{-}positive(FP) = \frac{\#\ mistakenly\ put\ on}{\#\ not\ about\ the\ topic} = \frac{b}{b+d}$, $false\text{-}negative(FN) = \frac{\#\ mistakenly\ missed}{\#\ about\ the\ topic} = \frac{c}{a+c}$. Figure 5 reports such a breakdown of classification errors for training and testing in the hierarchy. In the testing phase, a few lower level nodes resulted in relatively high FN. This is due to the depth of the nodes and the limited number of documents tested in those nodes.



(a) Training phase. (b) Testing phase.

Fig. 5. False-positive (fp) and false-negative (fn) errors (in percents).

5.1 Feature Subset Selection

The above experiment has shown that the TFIDF feature is a reliable indicator for categorizing Web text. We now investigate whether a subset of these features can perform as well in text classification.

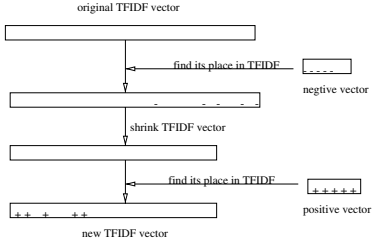
Straight TFIDF Subset One feature selection represents each node with only a subset of word features. The new feature vector for each node consists of the top m leading TFIDFs after sorting the original vector by its TFIDF values: $\boldsymbol{F}_m =< TFIDF(w_1, d), TFIDF(w_2, d), \ldots, TFIDF(w_m, d) >$, where $m <= |W|$.

Figure 6 illustrates the average accuracy (in percents) for classifiers at each depth for both training and testing phases. In testing, 10 different subsets are compared. According to our findings, when *TFIDFs* are cut down by 40-50%, they scarcely affect the test accuracy. This has two implications: First, not only is TFIDF a good indicator in text classification, but those higher TFIDFs values are the dominating terms. Second, using subset features can reduce TFIDFs storage requirements drastically and improve efficiency without compromising classification accuracy.

Special Positive/Negative Vectors Another way of selection is to introduce two special vectors (one positive, one negative) to represent the background knowledge. The positive and negative vectors are manually made. They are the human version of TFIDFs because most positive terms may coincide with the

		Average accuracy for classifiers at each depth of the hierarchy										
			Testing with a subset of Tfidf features									
		Training	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
d	1	97%	65%	70%	77%	81%	87%	93%	93%	93%	93%	94%
e	2	100%	57%	61%	68%	72%	78%	88%	88%	88%	90%	89%
p	3	90%	61%	72%	65%	75%	76%	79%	78%	78%	78%	76%
t	4	99%	57%	67%	68%	68%	68%	68%	68%	69%	70%	64%
h	5	98%	45%	55%	44%	45%	45%	46%	46%	48%	49%	32%

Fig. 6. Average accuracy for classifiers at each depth of the hierarchy.



(a) Design.

		training	testing
d	1	98%	96%
e	2	98%	92%
p	3	92%	82%
t	4	98%	63%
h	5	98%	53%

(b) Average accuracy.

Fig. 7. Combining background knowledge in the classification.

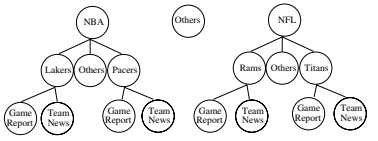
dominating TFIDFs, whereas negative terms are words that do not contribute to classification by human judgement.

Figure 7 (a) shows how this background knowledge is incorporated into the original TFIDFs for every node in the hierarchy. During training, the negative vector is subtracted from the original TFIDF vector. Later, in testing, we impose a confidence function and the positive vector on each node, and let the confidence function compete with the original similarity function. When background knowledge confidence reaches a certain level, we abandon the *cosine* measure and determine the category right away.

In this experiment, a very simple confidence function is used - the number of positive words (δ) that occur in the document. Such background knowledge somewhat mimics the the way a human determines a category. In Figure 7 (b), for example, when a δ of 4 is used, it reveals that combining such background knowledge into TFIDFs does improve the classification accuracy.

5.2 Applied to TV Closed Captions

We used the same approach as above, and applied it to TV closed caption data mixed with a few Web pages (15%). Closed captions usually contain more typos than Web pages. Even so, the classification accuracy showed some promising results (Figure 8). This is important because success in classifying TV closed captions can assist in video - or, generally multimedia - classification, in which case, time is of great concern.



(a) Concepts for TV closed caption data.

	training	testing	
		TFIDF w/ BK	BK alone
depth 1	96%	84%	81%
depth 2	91%	81%	39%
depth 3	90%	58%	20%

(b) Average accuracy.

Fig. 8. Classifying TV closed captions with TFIDF and background knowledge(BK).

6 Related Work

A number of existing approaches are similar to hierarchical classification based on a pre-defined concept hierarchy. Many of them are combined with feature subset selection which finds the best subset of features that improves classification accuracy, reduces measurement cost, storage, and computational overhead. One example, TAPER [1], makes use of a concept hierarchy and classifies text using statistical pattern recognition techniques. It finds feature subsets by the Fisher's discriminant. Similarly, Mladenic and Grobelnik proposed a document categorization method based on a concept hierarchy [8]. They used the naive Bayesian classifier on feature vector of word sequences and employed feature subsets to yield good performance. McCallum et al. also proposed a hierarchical classification using the naive Bayesian classifier [5]. In particular, they suggested combining labeled and unlabeled data to boost the classification accuracy.

There have been numerous approaches for automatic generation of a concept hierarchy. Their incentive is to eliminate the overhead of manually constructing a concept hierarchy. Sahami, for example, applied unsupervised clustering to generate a concept hierarchy from text data [10]. He made use of well-defined similarity measures to find the clusters as well as to feature subset selection. Sanderson and Croft presented a means of automatically deriving a hierarchical organization of concepts from a set of documents [12]. Their work used co-occurrence and subsumption conditions for selected salient words and phrases without standard learning or clustering techniques. In addition, there exist a variety of learning algorithms for text. Yang compared the performance of many learning algorithms [14]. Mladenic surveyed text-learning and related intelligent agents based on three key criteria: what representations to use for documents; how to select features, and what learning algorithm to use. Mitchell's book [6] is yet another comprehensive source of machine learning algorithms.

7 Summary and Discussion

The design of an intelligent text classifier is of great importance in the current world filled with such vast amounts of data. The algorithm must be fast because time is critical. To fulfill these promises, we designed and implemented a hierarchical classification system that leverages on a concept hierarchy and a simple and fast learning algorithm. The hierarchical aspect narrows down the search

space significantly by eliminating all irrelevant areas. Then, use of TFIDF and accuracy-feedback quickly makes the classifier.

Regarding experimental results, the hierarchical classifiers performed fairly well with the Web data and TV closed captions in the sports domain. Our preliminary work on feature subset selection also demonstrated improvements on classification accuracy and the cost associated with the use of features. Some avenues for future research include:

- Exploitation of data structures: Structural information in the text (e.g., title, sections, references) can be exploited and differentiated.
- Consideration of different types of data: The system can be tested with different types of text data - for instance, U.S. patent data.
- Incremental learning: New data can be learned dynamically. Meanwhile, old data should be ignored after some time has elapsed.
- Combination of labeled and unlabeled data: To reduce the overhead in data preparation prior to learning, unlabeled data can be combined with labeled data.
- Automatic expansion/shrinkage of the concept hierarchy: Dynamic change in the concept hierarchy is needed to accommodate the newly formed concepts.

References

1. S. Chakrabarti, B. Dom, R. Agrawal, P. Raghavan. Using Taxonomy, Discriminants, and Signatures for Navigating in Text Databases. In *Proceedings of the 23rd VLDB Conference*, 1997.
2. CNN.com. `http://www.cnn.com/`
3. M. Craven, D. DiPasquo, D. Freitag, A. McCallum, T. Mitchell, K. Nigam, S. Slattery. Learning to Extract Symbolic Knowledge from the World Wide Web. In *Proceedings of the 15th Conference on Artificial Intelligence*, 1998.
4. R. Korfhage. *Information Storage and Retrieval.* New York: Wiley, 1997.
5. A.McCallum, K. Nigam, J. Rennie, and K. Seymore. Building Domain-Specific Search Engines with Machine Learning Techniques. In *AAAI-99 Spring Symposium on Intelligent Agents in Cyberspace*, 1999.
6. T. Mitchell. *Machine Learning.* New York: McGraw Hill, 1997.
7. D. Mladenic. Text-learning and related intelligent agents: a survey. In *IEEE Intelligent Systems*, Vol.14, (no. 4), pages 44-54 July-Aug.1999.
8. D. Mladenic and M. Grobelnik. Featrure Selection for Classification based on Text Hierarchy. In *Working Notes of Learning from Text and the Web, Conference on Automated Learning and Discovery (CONALD)*, 1998.
9. US Patent and Trademark Office. `http://www.uspto.gov`
10. M. Sahami. *Using Machine Learning to Improve Information Access.* Ph.D. Dissertation, Department of Computer Science, Stanford University. 1998.
11. G. Salton. *Automatic Text Processing: the Transformation, Analysis, and Retrieval of Information by Computer.* Reading, Massachusetts: Addison-Wesley, 1989.
12. M. Sanderson, B. Croft. Deriving concept hierarchies from text. In *Proceedings of the 22nd ACM SIGIR Conference*, pages 206-213, 1999.
13. Yahoo. `http://www.yahoo.com/`
14. Y. Yang. A Re-examination of Text Categorization Methods. In *Proceedings of the 22nd ACM SIGIR Conference*, pages 42-49, 1999.

A Hybrid Technique for Data Mining on Balance-Sheet Data*

G. Dattilo[2], S. Greco[1,2], E. Masciari[1,2] and L. Pontieri[1,2]

[1] DEIS, Univ. della Calabria, 87030 Rende, Italy
[2] ISI-CNR, 87030 Rende, Italy
{dattilo,greco,masciari,pontieri}@si.deis.unical.it

1 Introduction

Recent rapid growth in the ability to generate and store data by more powerful Database Management Systems and hardware architecture, leads to a question: how can we take advantage of this large amount of information? Traditional methods for querying and reporting are inadequate because they can only manipulate data and the information content derived is very low. Obtaining new relationships among data and new hypotheses about them is the aim of Knowledge Discovery in Databases (KDD) which makes use of Data Mining techniques. These techniques have interesting applications for business data such as market basket analysis, financial resource planning, fraud detection and the scheduling of production processes. In this work we consider the application of Data Mining techniques for the analysis of the balance-sheets of Italian companies.

Currently there is a large interest in financial markets data mining and large amounts of data for this domain are available. Analysis of these data for the purpose of abstracting and understanding market behaviour are being intensively explored. These analysis are also useful for making predictions about future market evolutions and several companies use analytical data mining methods for actual investment portfolio management.

A number of Data Mining techniques e.g. rule induction, neural networks, and conceptual clustering, have been developed and used individually in domains ranging from space data analysis to financial analysis. Frequently, a single data mining technique is insufficient for extracting knowledge from a data set. Instead, several techniques must be employed cooperatively to support a single Data Mining application . In this paper we propose a hybrid Data Mining approach for the classification of balance-sheets. We show that by combining different classification techniques, such as induction of decision trees and clustering, it is possible to generate an appropriate partition and an intuitive description of data. We have implemented a system, called DMTool, based on the combined use of different Data Mining and classification techniques, for the analysis of balance-sheet data. Several experiments have confirmed the validity of our approach.

* Work partially supported by a MURST grant under the projects "Data-X" and "Piano Telematico Calabria" and by the EC project "Contact"

Y. Kambayashi, M. Mohania, and A M. Tjoa (Eds.): DaWaK 2000, LNCS 1874, pp. 419–424, 2000.
© Springer-Verlag Berlin Heidelberg 2000

2 Balance-sheet Classification

In this section we present an application which combines different classification techniques, such as induction of decision trees and Bayesian clustering, for the analysis of the balance-sheets of the Italian companies. Although we have considered several trades of the Italian economy, in this paper we only report results concerning the "Gum" trade.

Data Structure We have used a database containing the balance-sheets of all Italian companies. The most interesting data are stored in a table containing 520.000[1] tuples per year and each tuple contains 27 attributes (e.g. assets, turnover, profit, debt) describing the balance-sheet composition. First we have processed the data to eliminate redundant information and noise and selected the data used in our experiments as described below: 1) we have retained only data for the year 1998 (due to the presence of incomplete information for previous years); 2) we have considered only balance-sheets closed on December 31^{st} (for reason of uniformity); 3) we have analyzed different trades separately, since intrinsic differences in their behaviour would make any global analysis unreliable.

Moreover, we have also normalized data. More specifically, real values are scaled by a 10^6 factor whereas percentage values are truncated.

As mentioned above, in the experiments described in this paper, we focused the analysis on the "Gum" trade, which includes enterprises producing tyre and plastic etc.. This trade contains around 3.000 companies and, therefore, there are around 3.000 tuples whose *Asset* attribute (the one used by the domain expert to define the classes) spans a large range of values, from very low amounts to very high ones.

2.1 Combining Data Mining techniques

For the analysis of the balance-sheets we first used standard classification techniques, with the support of an expert for the definition of classes and, next, we combined induction of decision trees and Bayesian clustering to derive automatically, without the support of the domain expert, the classes definition.

The criteria suggested by the expert for the definition of classes consists in the partition of the domain of the attribute *Assets* as follows: Class 1: $Assets \leq 5$[2], Class 2: $5 < Assets \leq 10$, Class 3: $10 < Assets \leq 22$, Class 4: $22 < Assets \leq 38$, Class 5: $38 < Assets \leq 63$, Class 6: $63 < Assets \leq 140$ and Class 7: $Assets > 140$.

However, as we shall see, the description of the above classes is not satisfactory since more than the 12% of tuples are not classified correctly. Thus, we use Bayesian clustering to improve the partition suggested by the domain expert. This technique is based on the selection of the most relevant attributes (as suggested by the domain expert) and the application of the clustering technique in

[1] The number of Italian companies.

[2] Lit values scaled by a 10^6 factor

the following way: 1) generate a set of clusters by using the values of previously selected attributes; 2) memorize, for each cluster denoting a class, the associated tuples and delete selected attributes.

Clearly, if we consider all attributes we derive a set of disjointed classes (every tuple belongs to a unique class). However, in this case, generally, it is not possible to give a characterization of the classes in terms of intervals which are the attributes domain. Therefore, we prefer to derive a partition into classes less accurate but easily comprehensible. We introduce a technique which represents a good trade-off i.e. we search for a definition lesser accurate but easily understandable.

For each class C_j, each attribute A_i has associated a range R_{ij}. Since two ranges R_{ij} and R_{ik} may overlap, we select the attribute which minimizes overlaps as follows: 1) for each tuple we add the label provided by the clustering partition function; 2) we choose an attribute A' to characterize the clusters. In particular, we discretize A' by means of a supervised discretizing technique that uses the label assigned by the clustering model as target and for each tuple we choose the value obtained in the previous step as class label and remove the effective value of A'.

For choosing such an attribute we proceed as follows. For each attribute k and cluster C_i the clustering algorithm gives the average mean value($\mu_{i,k}$) and the standard deviation($\sigma_{i,k}$) of k within C_i. We define $R_{i,k}$ as:

$$R_{i,k} = [\mu_{i,k} - l * \sigma_{i,k}, \mu_{i,k} + l * \sigma_{i,k}]$$

where the value of l depends on the effective distribution of the attribute values within the class. We define an overlapping index as:

$$O_{i,j,k} = \frac{card(I_{i,j,k})}{card(C_i) + card(C_j)}$$

where $I_{i,j,k} = \{X \in C_i \cup C_j | \pi_k(X) \in R_{i,k} \cap R_{j,k}\}$ and $card(S)$ is the cardinality of the set S. We choose as target the attribute that minimizes the function: $F_k = \Sigma_i \Sigma_{j>i} O_{i,j,k}$

In our experiments the attribute that satisfies the above condition is *Assets* that induces the following partition: $Assets \leq 20$, $20 < Assets \leq 30$, $30 < Assets \leq 40$, $40 < Assets \leq 55$, $55 < Assets \leq 90$, $90 < Assets \leq 185$ and $Assets > 185$.

For reasons of uniformity we choose a partition into 7 classes characterized by a good accuracy, among the various ones obtained by clustering-based methods. This partition allows us to compare "homogeneous" experiments carried out on different class definition methods. Once we have defined a semantics for classes we use a *classifier* to build a model for data, containing a function for assigning objects to classes. Observe that the attribute used by the expert was the same attribute found by the clustering technique.

2.2 Experimental Results

We performed several experiments on different trades. Here we only report some results regarding the "Gum" trade.

For experimental purposes we used a training set having 1000 tuples. We first used the class partition suggested by a domain expert (an economic expert on balance-sheets). The results for this experiment are shown in Figure 1.

Pred. Cl.⇒	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7
Class 1	135	9	0	0	0	0	0
Class 2	8	122	13	1	0	0	0
Class 3	0	12	126	6	0	0	0
Class 4	2	1	18	116	7	0	0
Class 5	0	0	1	19	112	12	0
Class 6	0	0	0	0	8	132	4
Class 7	0	1	0	0	0	4	131

Fig. 1. Confusion matrix for domain expert partition

Numbers out of the diagonal correspond to *misclassified* tuples, namely tuples having an *effective* class (database class) different from the *predicted* class (founded by the algorithm). As an instance, in the second row we can note that "Class 2" (see previous section) is composed of 144 companies and 122 are correctly classified while the other 22 are assigned to neighboring classes. A quantitative expression for misclassification is the *generalization error* (i.e. the fraction of misclassified tuples with respect to the total number of tuples), which in this case is equal to 12.6%.

Pred. Cl.⇒	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7
Class 1	300	7	0	8	0	0	0
Class 2	8	281	2	0	0	0	0
Class 3	0	12	170	0	0	0	0
Class 4	9	0	0	117	0	4	0
Class 5	0	0	11	0	39	0	0
Class 6	4	3	0	1	0	10	0
Class 7	2	1	0	0	0	5	6

Fig. 2. Confusion matrix for clustering class partition

The same experiment using the class partition described in the previous subsection, gave the results reported in Figure 2. The generalization error is now 7.7%, less than in the previous experiment. After many experiments we observed that generalization errors obtained on clustering-based partitions are always less than the ones obtained on domain expert partitions.

3 A system for balance-sheet classification

In this section we describe the architecture and main functional aspects of a system, called *DMTool*, which we implemented to support the whole process of classification and data description of the balance-sheets of Italian companies.

DMTool provides an integrated environment for the management and classification of data as well as analysis of results. The system is built in a modular way, integrating several existing algorithms and packages. It allows inexpert users to elaborate data and build a model making use of data manipulation tools as well as clustering and decision tree induction algorithms. The system architecture is described in figure 3.

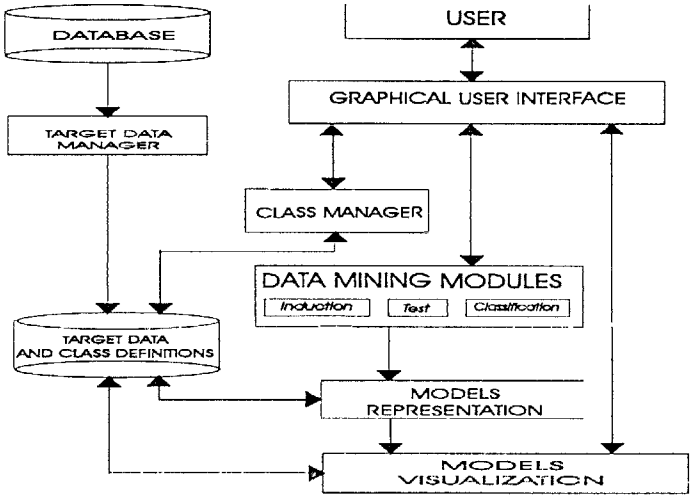


Fig. 3. System Architecture

Users interact with the system by means of a graphical user interface (GUI) that allows easy manipulation of original data in the source database. The system manages several archives: the source database, the target database, which contains data manipulated for a particular task, and a third database that stores models induced from data. In the last archive we store important information about model description and classification functions. For each model we store its representation, the estimated accuracy and information relative to the process which built the model (training-set, class definition, induction algorithm parameter). The main modules of the system are the following.

Target data management module This module is devoted to the selection of a subset of original data (*Target data*) for inducing a classification model. The main functions available include: tuple selection by two different criteria (random or by

user query), projection of interesting attributes and discretization for continuous numerical attributes.

Class management module This module offers easy and flexible functions for class definition. The main goal consists in defining a *target attribute* that will characterize object classes. We allow different ways to perform this operation so that users can choose the best method for the particular analysis they want to perform. In particular, we allow users to choose as target a discrete attribute, or to discretize a continuous one by using one of the several criteria available.

At the end of the classification process a report for misclassification is provided so that the user can verify the validity of the method chosen.

Data Mining Modules The main Data Mining component is the induction module. It permits the generation of a new classification model starting from the training set built by the user as described previously. The core of induction is an algorithm for induction of decision trees. The test module allows us to evaluate the prediction capabilities of an existing model applying it to a set of labeled test data, selected by the user. The result of this operation is a global measure of model accuracy and the confusion matrix. The confusion matrix permits us to estimate misclassifications and can suggest, if necessary, a more appropriate class definition for data, so the user can tune the method.

Visualization of data and extracted knowledge This module allows the graphical visualization of models as decision trees or decision rules ordered by class.

References

1. Mardia, K.V., Kent, J.T., Bibby, J.M.: Multivariant Analysis. Academic Press, New York (1979)
2. Chen, M.S., Han, J., Yu, P.S.: Data Mining: An Overview from a Database Perspective. In: IEEE Trans. on Know. Disc. and Data Eng. 8 (1996) 866–883
3. Cheeseman, P., Stutz, J. Bayesian Classification (Autoclass): Theory and Results. In: [6] (1996) 153–180
4. Dougherty, J., Kohavi, R., Sahami, M.: Supervised and unsupervised discretization of continuous features. In: Machine Learning, Proc. of the 12th Int. Conf. on Machine Learning. Morgan Kaufmann (1997) 194–202
5. Fayyad, U.M., Piatesky-Shapiro,G., Smyth, P.: From Data Mining to Knowledge Discovery: An overview. In [6] (1996) 1–36
6. Fayyad, U.M., Piatesky-Shapiro,G., Smyth, P. (eds): Advances in Knoweldge Discovery and Data Mining. The MIT Press, Menlo Park (1996)
7. Quinlan, J.R.: Induction of Decision Trees. Machine Learning 1 (1986) 81–106
8. Salzberg, S.L.: On Comparing Classifiers: Pitfalls to Avoid and a Recommended Approach. Data Mining and Knowledge Discovery 1 (1997) 317–328
9. Scheffer, T., Herbrich, H.: Unbiased assessment of learning algorithm. In: Proc. 15th Int. Joint Conf. on Artif. Intell. Morgan Kaufmann (1997) 798–803
10. Hanson, R., Stutz, J., Cheeseman, P.: Bayesian classification with correlation and inheritance. In: Proc. 12th Int. Joint Conf. on Artif. Intell. Morgan Kaufmann (1991) 692-698

Mondou: Information Navigator with Visual Interface

Hiroyuki Kawano[1] and Minoru Kawahara[2]

[1] Department of Systems Science, Kyoto University, Kyoto 6068501, Japan
kawano@i.kyoto-u.ac.jp
WWW home page: http://www.kuamp.kyoto-u.ac.jp/~kawano/index.html
[2] Data Processing Center, Kyoto University, Kyoto 6068501, Japan
kawahara@kudpc.kyoto-u.ac.jp
WWW home page: http://www.kudpc.kyoto-u.ac.jp/~kawahara/index.html

Abstract. Since 1995, we have been developing web search engine "Mondou" using data mining techniques (http://www.kuamp.kyoto-u.ac.jp/labs/infocom/mondou/index_e.html) in order to discover the helpful information for web search operations. In our previous works, we focus on the computing cost to derive associative keywords, we propose the method to determine system parameters, such as $Minsup$ and $Minconf$ threshold values. Moreover we evaluate the ROC performance of derived keywords by weighted association algorithms. In this paper, we try to implement two kinds of Java applets in our **Mondou** system, such as ROC graph for selecting associative keywords and documents clustering. This visual interface shows characteristics of associative rules on the ROC graph with the $Minsup$ values. It also provides the function of document clustering in order to visualize retrieved documents.

1 Introduction

In the research fields of data mining[10, 2] and text/web mining [1], various mining algorithms have been proposed to discover interesting patterns, rules, trends and representations in databases. Almost of discovered patterns or rules are rather simple as knowledge for experts, but they are sometimes helpful for beginners, who don't have background or domain knowledge. Therefore, we extended the algorithm of association rules[14], and we implemented our proposed algorithm in our web search engine "Mondou" (http://www.kuamp.kyoto-u.ac.jp/labs/infocom/mondou/). The Mondou system provides associative keywords to search users by extended association algorithms[9].

When we developed our web mining engine, we had several problems to derive appropriate associative keywords. Since the quality of derived keywords strongly depends on the minimum support threshold ($Minsup$) and the minimum confidence threshold ($Minconf$). Therefore, in our previous works[4], we proposed the way to adjust the $Minsup$ value dynamically.

In this paper, based on our previous works[5, 6], we introduce a method which specifies the optimal thresholds based on the ROC (Receiver Operating Characteristic) analysis[11]. In Section 2, we show the simple sketch of performance

Y. Kambayashi, M. Mohania, and A M. Tjoa (Eds.): DaWaK 2000, LNCS 1874, pp. 425–430, 2000.
© Springer-Verlag Berlin Heidelberg 2000

evaluation of our proposed algorithm, which is implemented in Mondou system. In Section 3, we present the implementation of visual interface for Mondou system. This interface is based on the characteristics of ROC graph and threshold values. Furthermore, we show the clustering interface of Mondou system. Finally, we make concluding remarks and discuss the future works in Section 4.

2 ROC Evaluation of associative keywords

In this section, we show the way to estimate the performance of derived keywords by our **Mondou** search engine. We define following parameters:

Definition:

$\mathbf{G}$: A set of keywords including in a query
n: The number of keywords in $\mathbf{G}$
k_i: The i'th keyword in $\mathbf{G}$ $(1 \leq i \leq n)$
$\mathbf{K}_i$: The set of documents covered by k_i
$\mathbf{B}$: The set of documents covered by $\mathbf{G}$
m: The number of keywords derived from item set $\mathbf{B}$
r_j: The j'th keyword derived from item set $\mathbf{B}$ $(1 \leq j \leq m)$
$\mathbf{R}_j$: The set of documents covered by r_j

Moreover, we use the following operators: $\bigcup$ is the set operator of union, $\bigcap$ is the set operator of intersection, and $||$ is the set operator to count the number of items.

Figure 1 shows a status of coverage by $\mathbf{B}$ and $\mathbf{R}_j$ in the universal set $\mathbf{U}$, which means all database items in the system. The items covered by all keywords in $\mathbf{G}$ is $\mathbf{B} = \bigcap_{i=1}^{n} \mathbf{K}_i$.

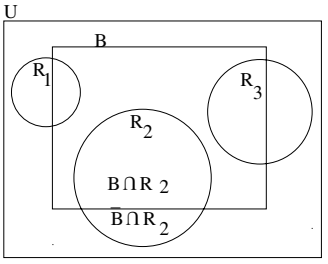
Next, in order to evaluate the performance of derived keywords r_j, we introduce the ROC analysis[11] [1]. An instance can be classified into two classes: the positive class P or the negative class N, and positive y (yes) or negative n (no) are assigned by a classifier. We define that $p(c \mid I)$ is the posterior probability that instance I is positive c. The true positive rate TP of a classifier is the following equation:

$$TP = p(y \mid P) \simeq \frac{\text{positive correctly classified}}{\text{total positives}}. \tag{1}$$

The false positive rate FP of a classifier is as follows:

$$FP = p(y \mid N) \simeq \frac{\text{negative incorrectly classified}}{\text{total negatives}}. \tag{2}$$

[1] ROC analysis has been used in the signal detection theory to depict tradeoffs between the hit rate and the false alarm rate. ROC graphs illustrate the behavior of a classifier without regard to class distribution or error cost, they decouple classification performance from these factors. Moreover, the ROC convex hull method is an effective way to compare multiple classifiers, it specifies the optimal classifier with the highest performance.



U: The universal set of database items
B: A set of items covered by a query (G)
R_j: A set of items covered by a associative rules derived from B, where j=1,2,...m.

Fig. 1. Status of database items covered by keywords.

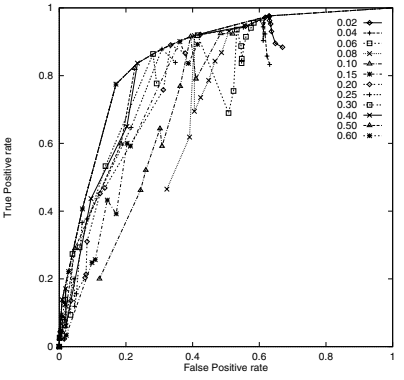


Fig. 2. The ROC graph of *Minsup*.

Therefore, in our retrieved set, the positive instance is $\mathbf{B}$ that decreases the number of documents and the negative instance is $\overline{\mathbf{B}}$ that increases the number of them. Thus the true positive instance is $\mathbf{B} \cap \bigcup_{j=1}^{m} \mathbf{R}_j$, and the false positive instance is $\overline{\mathbf{B}} \cap \bigcup_{j=1}^{m} \mathbf{R}_j$. TP and FP are represented by the following equations.

$$TP = \frac{\mid \mathbf{B} \cap \bigcup_{j=1}^{m} \mathbf{R}_j \mid}{\mid \mathbf{B} \mid}, \; FP = \frac{\mid \overline{\mathbf{B}} \cap \bigcup_{j=1}^{m} \mathbf{R}_j \mid}{\mid \overline{\mathbf{B}} \mid}$$

In Figure 1, the true positive instance is $\mathbf{B} \cap \mathbf{R}_2$ and the false positive instance is $\overline{\mathbf{B}} \cap \mathbf{R}_2$.

Furthermore, we need to control the value of $Minsup$ according to the frequency of the keywords appeared in the database. Therefore, we try to divide into several classes by the frequency of keywords, and we plot averages of (FP, TP) derived from specific keywords, considering the categories in an ROC graph.

For instance, in order to evaluate the performance of our proposed algorithm, we analyzed 330,562 documents, $\mid \mathbf{U} \mid$, in Mondou system, and divided ten categories. Figure 2 shows the ROC graph plotted by the averages of (FP, TP) for each category and each $Minsup$ with the boundary of the convex hull. We can adjust $Minsup$ value dynamically based on the characteristics in Figure 2. Furthermore, if we draw the distribution of associative keywords with a few attributes, it should be helpful to determine the combination of keywords in a modified query.

3 Visual Interface of Mondou

At present, we have been constructing the web-based information navigating interface using CGI and the "Open Text" database. In Figure 3, we show the query window of Mondou system in [8, 3]. In addition to these primitive interfaces, we try to implement two kinds of Java applets in Mondou system.

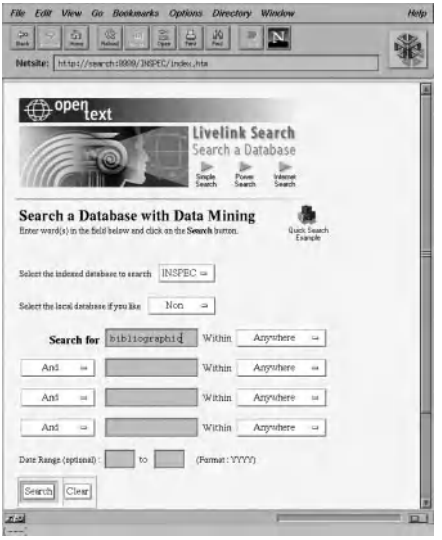


Fig. 3. Query window of Mondou.

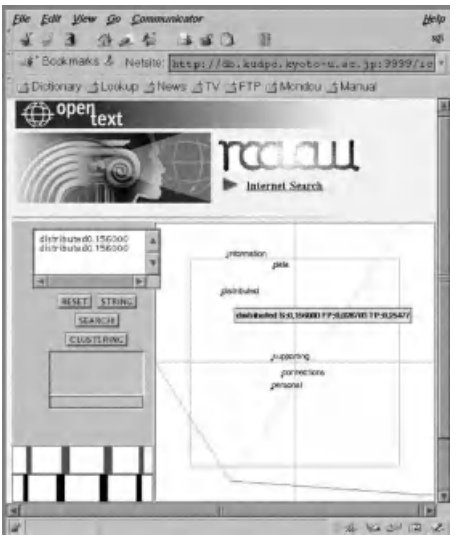


Fig. 4. Visual interface of keyword selection.

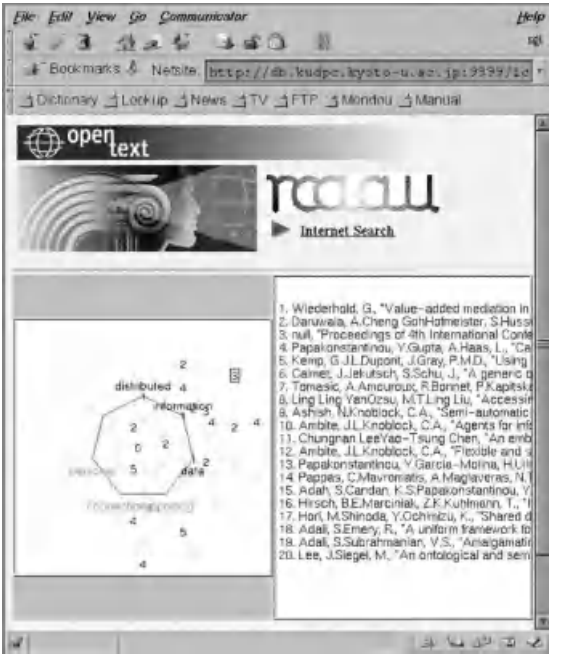


Fig. 5. Clustering results by Mondou.

According to the discussion in our previous paper[7] and in the previous section, as the distance from the point $(1, 0)$ in the original ROC graph becomes longer, the performance of keywords becomes higher. Therefore, we have to visualize the distance of it by Java applet. By using ROC graph with $(FP, TP, Minsup)$, we can choose much more effective keywords easily. Thus, in order to select keywords, we implemented the ROC 3D graph with (FP, TP) and $Minsup$ values, which is shown in Fig.4. Moreover, in this applet, we also implemented several functions of rotating, magnifying and fisheye[12] in order to inspect the ROC graphs.

For example, we executed the query with a keyword "*mediator*" in Fig. 3, Mondou system derives several associative keywords, such as { *information, data, distributed, supporting, connections, personal* } with appropriate $Minsup$ values, which is shown in Fig.4. By using this 3D ROC graph, we understand the effectiveness of associative keywords in order to make clusters of retrieved documents. After selecting keywords in Fig.4, we try to execute "*documents clustering*" functions by the left side button.

By visualization of document clusters which is based on VIBE system [13], in Fig.5, we easily understand the items in target clusters including appropriate documents without complicated query expressions of several keywords. For instance, when you select the number in Fig.5, the items in the cluster are shown automatically in the right frame, these results satisfy the "**AND**" expression of keywords "*mediator, distributed, information, data*".

In the results, the former visualization applet is very powerful for selecting suitable keywords by ROC graph and support values. The latter visual interface is effective for clustering of retrieved documents by combinations of selected keywords. These visual interfaces are very helpful tools for information navigation Mondou with mining association rules.

4 Conclusion

The technique of information visualization becomes to be used widely in various data mining researches. However it is too hard to determine the effective methods. In this paper, at first, we evaluate the performance of rules based on the characters of ROC graphs. Then, considering the ROC convex hull method, we try to select appropriate combination of keywords with suitable threshold values on the Browser. Moreover, we try to make clusters for retrieved documents. In the future, in order to provide much more easy query operations, we have to develop much more sophisticated visual applets in our Mondou system.

Acknowledgment

A part of this work is supported by the grant of Scientific Research (11130211) from the Ministry of Education, Science, Sports and Culture of Japan. We thank Nissho Iwai Infocom Co., Ltd. for the support of "OpenText" Database.

References

1. Feldman, R.: *Practical Text Mining*, Second Symposium on Principles of Data Mining and Knowledge Discovery (PKDD-98), Nantes, France (1998).
2. Han, J., Nishio, S., Kawano, H. and Wei, W.: *Generalization-based Data Mining in Object-oriented Databases Using an Object Cube Model*, Data and Knowledge Engineering, No.25, pp.55-97 (1998).
3. Kawahara, M., Kawano, H. and Hasegawa, T.: *Data Mining Technologies for Bibliographic Navigation System*, Transactions of the IPSJ, Vol.39, No.4, pp.878–887 (1998).
4. Kawahara, M., Kawano, H. and Hasegawa, T.: *Implementation of Bibliographic Navigation System with Text Data Mining*, Systems Science Vol.24, No.3, pp.101-118 (1998).
5. Kawahara, M. and Kawano, H.: *Performance Evaluation of Bibliographic Navigation System with Association Rules from ROC Convex Hull Method*, Transactions of the IPSJ:Database, Vol.40, No.SIG3(TOD1), pp.105–113 (1999).
6. Kawahara, M. and Kawano, H.: *Mining Association Algorithm Based on ROC Convex Hull Method in Bibliographic Navigation System*, Proc. of Second International Conferences DS'99, LNAI 1721, Tokyo, pp.333-334, (1999).
7. Kawahara, M. and Kawano, H.: *Performance Evaluation and Visualization of Association Rules using Receiver Operating Characteristic Graph*, 1999 International Symposium on Database Applications in Non-Traditional Environments, DANTE'99, Kyoto, pp.333-342, (1999).
8. Kawahara, M. and Kawano, H.: *An Application of Text Mining: Bibliographic Navigator Powered by Extended Association Rules*, Proc. of Hawaii International Conference on System Sciences (HICSS-33), p.34 (CD-ROM), Hawaii, (2000).
9. Kawano, H.: *Mondou: Web search engine with textual data mining*, Proc. of IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, pp.402–405, (1997).
10. Michalski, R.S., Bratko, I. and Kubat, M. (eds.): *Machine Learning and Data Mining, Methods and Applications*, John Wiley & Sons, (1998).
11. Provost, F. and Fawcett, T.: *Analysis and Visualization of Classifier Performance: Comparison under Imprecise Class and Cost Distributions*, Proc. of 3rd Int'l Conf. on Knowledge Discovery and Data Mining (KDD-97), pp.43-48, (1997).
12. Sarkar, M., Brown, M.H.: *Graphical Fisheye Views*, Communications of the ACM, pp.73-84, Dec., (1994).
13. Korfhage, R.R.: *To see, or not to see– is That the query?*, Proc. of the 14th SIGIR, pp.134-141, Chicago, Oct., (1991),
14. Srikant, R. and Agrawal, R.: *Mining Generalized Association Rules*, Dayal, U., Gray, P. M. D. and Nishio, S. (Eds.), Proc.21st VLDB, pp.407–419, Zurich, Switzerland (1995).

Vmhist: Efficient Multidimensional Histograms with Improved Accuracy

Pedro Furtado and Henrique Madeira
CISUC/DEI

Abstract: Data warehouses must be able to process and analyze large amounts of information quickly and efficiently. Small summaries provide a very efficient way to obtain fast approximate answers to complex queries that run for too long. This paper proposes an efficient hierarchical partitioning strategy *vmhist* achieving a large improvement in the accuracy of the summary while maintaining all scalability. This is achieved by pre-computation, localized updating and additivity of the error measures used in the partitioning process. Evaluation reveals that a significant accuracy improvement is obtained for summaries produced with *vmhist* without significant increase in histogram construction time cost.

1. Introduction

Multidimensional histograms provide a very practical way to produce small queriable summaries of the joint distribution of values of multidimensional data sets such as those used in OLAP. The most important metrics to evaluate the quality of algorithms used for the construction of histograms are the accuracy of the summary when answering queries and the scalability of the algorithm (the algorithm must be useful to reduce very large data sets). In fact, the scalability of the algorithm and the accuracy are indirectly related, as the most accurate summary construction algorithms are frequently not scalable.

The *mhist* algorithm was proposed in [3] for the construction of multidimensional histograms for selectivity estimation and in [5] for approximate OLAP query answering. This algorithm proposes a judicious way to construct histograms by using the marginal distributions to determine the partitioning into buckets instead of doing a more time-consuming analysis (the marginal distributions are small comparing to the data set size). The space partitioning metric used in [3] was based on the determination of the largest differences between neighboring values in the marginal distribution (*mhist-Maxdiff*). It is well-known that a heuristic based on the homogeneity of the resulting buckets, such as the V-optimal variance based constraint [4] would produce higher quality histograms, but this heuristic would imply exponential construction time and therefore scalability would be lost, rendering the approach unusable [3,4]. Recently, this problem was dealt with for the V-Optimal constraint on uni-dimensional histograms for data sets with a reasonably small cardinality of values [1]. The problem of finding a scalable high-quality solution for high cardinality multi-dimensional data which would provide an important improvement to the *mhist* algorithm remained unsolved.

The main contribution of this paper is to solve this problem by proposing a histogram construction strategy *vmhist* that is able to maintain scalability while using the variance of candidate buckets to conveniently probe the homogeneity of regions, and therefore produce a high-quality multidimensional histogram without extra

Y. Kambayashi, M. Mohania, and A M. Tjoa (Eds.): DaWaK 2000, LNCS 1874, pp. 431–436, 2000.
© Springer-Verlag Berlin Heidelberg 2000

performance cost. The algorithm uses pre-computation of values, additivity of error measures and localized updating features to achieve very accurate summaries while maintaining scalability.

The paper is organized as follows: section 2 introduces hierarchical partitioning and section 3 presents the *vmhist* technique. Section 4 presents experimental results and section 5 concludes the paper.

2. Hierarchical Partitioning

In order to produce a histogram summary of a data set, a space partitioning strategy must be followed. Adaptable partitioning refers to strategies that try to determine the best possible partitioning of the space into regions [2]. This section describes the basic hierarchical partitioning algorithm for multidimensional spaces. Given a data set with point values p(a1,...,an,v1,...,vj,…,vm), the marginal data distribution of attribute ai for the value attribute vj is the set marg_ai(xl, Σall database tuples with ai = xl vj), where xl represents each individual value taken by attribute ai. The marginal distribution is used in the analysis that determines the partitioning instead of the whole data, with obvious performance gains. The algorithm successively divides one or more buckets into two by the end-to-end splitting line that gives the best error measure results.

Figure 1 shows a two-dimensional space with the arrays of marginal distributions (sum of values or occurrences in the row or column) which are analyzed to determine the splitting lines.

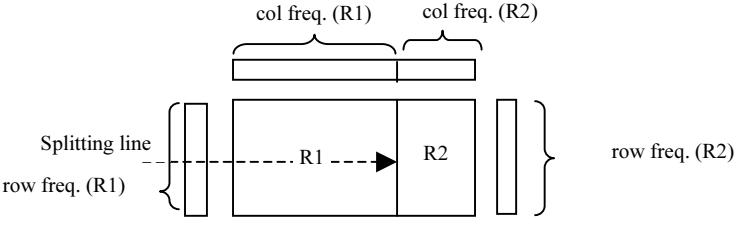


Figure 1- Hierarchical Partitioning and Marginal Distribution Arrays

Alternative partitioning constraints can be used to determine the best splitting line from the marginal distributions. For Maxdiff histograms (mhist-Maxdiff) [3], one with the largest difference in source values between adjacent values is used. For V-Optimal (variance optimal) histograms [4], the splitting line with maximum variance of source parameter values is used.

A naive implementation of a variance-based constraint in multidimensional data sets incurs in exponential histogram construction time due to the need to re-compute the measure in each iteration using the individual values of partitioned regions.

The next section proposes the *vmhist* technique, which uses pre-computed additive parameters and localized updates to render variance-based constraints useful in the multidimensional data set context.

3. Variance-Based Hierarchical Partitioning (*vmhist*)

The technique v*mhist* proposes a new scalable strategy to implement complex variance-based heuristics within the multidimensional partitioning context. The key concepts of this algorithm are:

- **Pre-computation -** almost all the parameters that are needed in the iterative partitioning process are pre-computed in extended marginal distribution arrays;
- **Additivity -** the parameters that are pre-computed and used in the computation of the homogeneity of regions are additive. This way, it is possible to compute the homogeneity without accessing all the values from the regions. Instead, the pre-computed parameters are used to derive a variance measure. Furthermore, only the region that was split needs an update of the homogeneity measure;
- **Localized updates** - only a very small set of the pre-computed values, corresponding to the region(s) that were split, have to be updated in each iteration.

The algorithm *vmhist* applies the policies described above to the basic hierarchical partitioning strategy. In particular, the variance can be computed additively for any region B_i from the values n_{Bi} (nº of values), ls_{Bi} (linear sum of values) and ss_{Bi} (square sum of values), as shown next:

$$\text{Variance} = \frac{\Sigma_{i=1}^{n} (v_i - \mu)^2}{n}$$

$$\text{Additivity: } V_{Bi}(n_{Bi}, ss_{Bi}, ls_{Bi}) = (ss_{Bi} - ls_{Bi}^2)/ n_{Bi}$$

3.1. Pre-computing and Localized Updating of Partitioning Parameters in *vmhist*

The split decision in *mhist*-MaxDiff [3] is taken locally (neighbor marginal distribution values). In contrast, the variance-based approach proposed here probes the variance resulting from all possible hierarchical splits in all dimensions to determine the split that produces the highest overall reduction in variance in each iteration. After the split, only the variances of the two resulting buckets need to be recalculated and this is done efficiently using pre-computed values. This means that *vmhist* uses the variance measure without the need to re-compute the values of the partitioning constraint except locally. This approach is only possible because the marginal distribution contains all the values that are needed to calculate the variance without accessing the source table.

As we have shown, the variance is easily expressed as a function of the additive parameters n_{Bi}, ss_{Bi} and ls_{Bi}. It can be computed for the union of any number of regions using these parameters that must be available for each region:

$$V_{Bi \cup Bj \cup \ldots \cup Bl} = V(n_{Bi} + \ldots + n_{Bj}, ls_{Bi} + \ldots + ls_{Bj}, ss_{Bi} + \ldots + ss_{Bj})$$

This way, the pre-computed quantities that must be kept to speedup the computation of variance are simply the additive parameters for each position of the marginal distributions, which have the structure:

dimension	// identifies the dimension attribute
bucket	// identifies the region
a	// value taken by the dimension attribute

$count_i$	// number of occurences
ls_i	// sum of values
ss_i	**// square sum of values**
mv_i	// the splitting condition to be maximized

The statement that creates the marginal distributions becomes:

SELECT count(d_i) , sum(v_i), **sum($v_i * v_i$)**
FROM sourcetable
GROUP BY d_i;

Figure 2 illustrates the extended marginal distribution with the parameters described above and also additional cumulative sums that are discussed next.

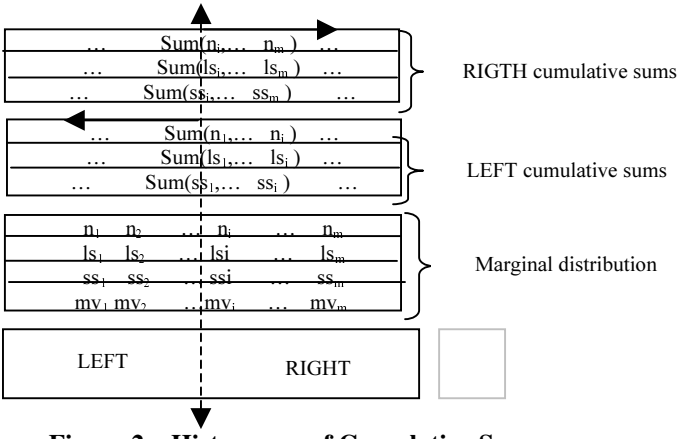


Figure 2 – Histograms of Cumulative Sums

The pre-computation and localized updating strategy is modified further in *vmhist*, to render the computation of the variance constraint extremely fast. The variances of the buckets formed before and after each candidate splitting line can be computed at any moment from pre-computed cumulative sums of the parameters that are used to derive the variance additively. The structure used to store the pre-computed parameters is a cumulative histogram that computes the sums of ls, ss and count, as shown in Figure 2. The variance and the cumulative values are both pre-computed in the beginning and only updated locally in each iteration. The LEFT/RIGHT histograms are pre-computed or updated by summing the values from the marginal distributions to the left or right of each value. For instance, the following statement updates the LEFT histogram:

sum($count_j$), sum(ls_j) , sum(ss_j), (sum(ss_j) - sum(ls_j) × sum(ls_j) /sum($count_j$)
FROM marginal distribution
WHERE a_j <= a_i.

3.2. The Variance-based Partitioning Constraint of *vmhist*

In *vmhist*, a simple variance-based partitioning constraint was used which achieves very good accuracy results. The algorithm determines the best splitting lines as those with largest decrease in variance between the region and the candidate subregions for each possible splitting line. The *mv* parameter of the marginal distribution arrays is pre-computed and updated locally. The partitioning constraint maximizes the difference of variances between the bucket and the two buckets resulting from a possible partitioning:

$$\text{Variance(B)} - (\text{Variance(B1)} + \text{Variance(B2)})$$

$$(\ mv = \text{B}.n \times \text{B.var} - \text{left}.n \times \text{left.var} - \text{right}.n \times \text{right.var}\ )$$

The best splitting line(s) are computed by maximization of the *mv* parameter.

4. Experimental Results

The main issues in the comparative evaluation between histogram construction algorithms are the construction time cost and the accuracy of the resulting summary. The algorithms must be scalable, as they are intended to summarize very large data sets. Figure 3 shows the construction time cost of three strategies: *mhist-Maxdiff* [3] (the basic implementation of *mhist* using the Maxdiff partitioning constraint); *vmhist* implementation of the Maxdiff (*vmhist-Maxdiff*), which also precomputes the partitioning constraint, and variance variant of vmhist (*vmhist-Variance*). The results show that the pre-computation and localized update features of *vmhist* resulted in faster execution than the basic *mhist* implementation that needs to re-compute the partitioning constraint in each iteration. The results also show that *vmhist-Variance* is practically as fast as *vmhist* using the Maxdiff constraint. In fact, it can be seen from the figure that the *vmhist* strategy is more scalable than *mhist-Maxdiff*.

Figure 4 shows comparative accuracy results between *mhist-Maxdiff* and *vmhist*, considering range-sum aggregation queries on a Sales data set and summaries with 5% of the original data set size. The aggregation categories are shown in the *x*-axis and the average number of values per group result are also shown. It is important to note that the accuracy is very good for any summary when large ranges are considered in all dimensions of the data set, but much lager errors result when queries detail some dimensions further or select fewer values. The errors of the group results are measured relative to the average value returned by the query. The figure shows that *vmhist* was always able to produce much better estimations than *mhist*.

Conclusions and Future Work

This paper presents a new hierarchical partitioning summary construction technique. This technique uses error measure additivity, pre-computation and localized updates to derive very accurate summaries. These optimization features allow variance constraints to be used scalably to summarize very large data sets,

providing a significant improvement over the previous strategies. The experimental results show the scalability and improved accuracy of the technique.

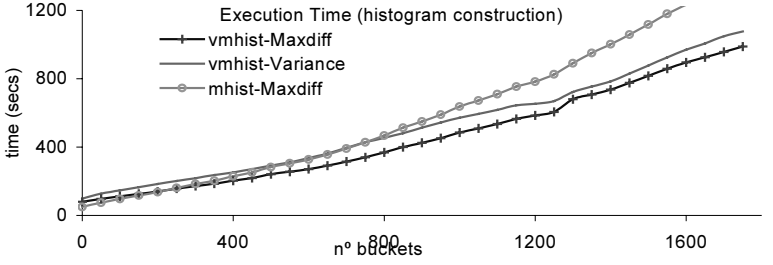


Figure 3–Histogram Construction Cost

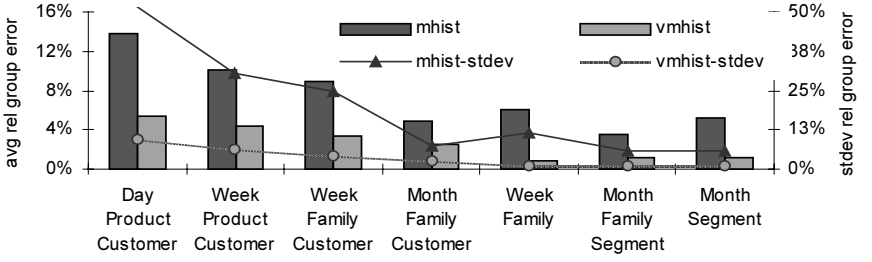


Figure 4-Comparative Accuracy of *vmhist* and *mhist*

References

1. H.V. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K. Sevcik, and T. Suel, "Optimal Histograms with Quality Guarantees", in Proc. 24th International Conference on Very Large Data Bases, August 1998, pp. 275-286.
2. S. Muthukrishnan, V. Poosala and T. Suel. "On Rectangular Partitionings in Two Dimensions: Algorithms, Complexity and Applications", in Procs. of the International Conference on Database Theory, Jerusalem, 1999.
3. V. Poosala, Y. Ioannidis, "Selectivity Estimation Without the Attribute Value Independence Assumption", Proceedings of the 23rd VLDB Conference, Athens, Greece, 1997.
4. V. Poosala, "Histogram-Based Estimation Techniques in Database Systems". PhD Thesis, University of Winsconsin – Madison, 1997.
5. V. Poosala, V. Ganti, "Fast Approximate Answers to Aggregate Queries on a Data Cube". 11th International Conference on Scientific and Statistical Database Management, Cleveland, Ohio, USA, 1999. IEEE Computer Society.
6. Special Issue on Data Reduction Techniques of the Bulletin of the Technical Committee on Data Engineering of the IEEE Computer Society, December 1997, Vol. 20, n 4.
7. T. Zhang, R. Ramakrishnan and M. Livny, "BIRCH: an Efficient Data Clustering Method for Very Large Databases", Proc. 1996 SIGMOD.

Author Index

Aboa J.-P. 393
Agrawal D. 24, 67, 77
Albrecht J. 47

Bae J. 276
Barbará D. 234
Bauer A. 47, 189
Bellatreche L. 57
Ben-Eliyahu-Zohary R. 265
Berchtold S. 244
Bhargava B. 287
Böhm C. 244, 372
Bouzeghoub M. 178
Braunmüller B. 372

Cabibbo L. 1
Calders T. 399
Chen Q. 210
Chen X. 327
Chuang W.T. 409
Chung S.M. 290

Dattilo G. 419
Davidson S.B. 114
Dayal U. 210
Ding L. 104

El Abbadi A. 24, 67, 77
Emilion R. 393
Ezeife C.I. 126

Farach-Colton M. 358
Fu A.W.-c. 317
Furtado P. 162, 431

Giuffrida G. 409
Goethals B. 307
Golfarelli M. 11
Greco S. 419
Gudes E. 265

Han H.-S. 337
Han J. 347

Holt J.D. 290
Hotho A. 258
Hristescu G. 358
Hsu M. 210
Hsu W. 337
Hümmer W. 189
Huynh T.N. 83

Kam P.-s. 317
Karlapalem K. 57
Kawahara M. 425
Kawano H. 425
Kedad Z. 178
Keim D.A. 244
Kim E.D. 347
Kitsuregawa M. 301
Koeller A. 168
Kotsis N. 146
Kriegel H.-P. 244, 372

Lam J.M.W. 347
Lee S. 276
Lehner W. 189
Li Q. 57
Liefke H. 114
Liu B. 337

Madeira H. 162, 431
Maedche A. 258
Maio D. 11
Masciari E. 419
McGregor D.R. 146
Mercy J.-L. 134
Morzy T. 382

Nguyen B.T. 83
Niemi T. 199
Nummenmaa J. 199

O'Gorman K. 77

Pajarola R. 24
Paredaens J. 399

Petrounias I. 327
Pontieri L. 419
Pramudiano I. 301

Redert M. 47
Riedewald M. 24
Rizzi S. 11
Rundensteiner E.A. 94, 104, 168

Sapia C. 224
Schiefer J. 83
Schrefl M. 34
Sonnberger H. 134
Stanoi I. 67

Thalhammer T. 34
Thanisch P. 199
Tiyyagura A. 409
Tjoa A M. 83
Torlone R. 1

Van den Bussche J. 307

Wiese M. 258
Wojciechowski M. 382
Wu X. 234

Xia Y. 337
Xu M. 126
Xu X. 244

Yang J. 409
Yoshizawa T. 301

Zakrzewicz M. 382
Zhang X. 94, 104