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

Yahiko Kambayashi Werner Winiwarter
Masatoshi Arikawa (Eds.)

Data Warehousing and Knowledge Discovery

Third International Conference, DaWaK 2001
Munich, Germany, September 5-7, 2001
Proceedings

Springer

Volume Editors

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

Werner Winiwarter
EC3
Siebensterngasse 21/3, 1070 Wien
E-mail: werner.winiwarter@ec3.at

Masatoshi Arikawa
University of Tokyo, Center for Spatial Information Science (CSIS)
4-6-1, Komaba, Meguro-ku, Tokyo 153-8904, Japan
E-mail: arikawa@csis.u-tokyo.ac.jp

Cataloging-in-Publication Data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Data warehousing and knowledge discovery : third international conference ; proceedings / DaWaK 2001, Munich, Germany, September 5 - 7, 2001. Yahiko Kambayashi ... (ed.). - Berlin ; Heidelberg ; New York ; Barcelona ; Hong Kong ; London ; Milan ; Paris ; Tokyo : Springer, 2001
(Lecture notes in computer science ; Vol. 2114)
ISBN 3-540-42553-5

CR Subject Classification (1998): H.2, H.3, C.2, H.4, H.5, J.1

ISSN 0302-9743
ISBN 3-540-42553-5 Springer-Verlag Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by PTP Berlin, Stefan Sossna
Printed on acid-free paper SPIN 10839833 06/3142 5 4 3 2 1 0

Preface

Data Warehousing and Knowledge Discovery technology is emerging as a key technology for enterprises that wish to improve their data analysis, decision support activities, and the automatic extraction of knowledge from data.

The objective of the Third International Conference on Data Warehousing and Knowledge Discovery (DaWaK 2001) was to bring together researchers and practitioners to discuss research issues and experience in developing and deploying data warehousing and knowledge discovery systems, applications, and solutions.

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 association rules, mining temporal patterns, data mining techniques, collaborative filtering, Web mining, visualization, matchmaking, evelopment and maintenance of data warehouses, OLAP, and distributed data warehouses. 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 34 papers. The conference program included one invited talk: "Knowledge Management in Heterogeneous Data Warehouse Environments" by Professor Larry Kerschberg, George Mason University, USA.

We would like to thank the DEXA 2001 Workshop General Chair (Professor Roland Wagner) and the organizing committee of the 12th International Conference on Database and Expert Systems Applications (DEXA 2001) for their support and their cooperation. Many thanks go to Ms Gabriela Wagner for providing a great deal of help and assistance as well as to Mr Bernd Lengauer for administrating the conference management software. We are very indebted to all program committee members and additional reviewers who have reviewed the papers very carefully and timely. We would also like to thank all the authors who submitted their papers to DaWaK 2001; they provided us with an excellent technical program.

September 2001

Yahiko Kambayashi
Werner Winiwarter
Masatoshi Arikawa

Program Committee

General Chair:
Yahiko Kambayashi, Kyoto University, Japan

Program Chairs:
Werner Winiwarter, Software Competence Center Hagenberg, Austria
Masatoshi Arikawa, University of Tokyo, Japan

Program Committee Members:
Tatsuya Akutsu, University of Tokyo, Japan
Hiroki Arimura, Kyushu University, Japan
Mike Bain, The University of New South Wales, Australia
Elena Baralis, Politecnico di Torino, Italy
Roberto Bayardo, IBM Almaden Research Center, USA
Anjali Bhargava, TRW, USA
Bharat K. Bhargava, Purdue University, USA
Stefan Berchtold, stb software technology beratung gmbh, Germany
Ulrich Bodenhofer, Software Competence Center Hagenberg, Austria
Christian Böhm, Ludwig-Maximilians-University of Munich, Germany
Luca Cabibbo, Università degli Studi Roma Tre, Italy
Murray Campbell, IBM TJ Watson Research Center, USA
Tiziana Catarci, Università degli Studi di Roma "La Sapienza", Italy
Peter Chamoni, Gerhard-Mercator-University Duisburg, Germany
Arbee L. P. Chen, National Tsing Hua University, Taiwan
Janet Chen, Philips Research East Asia, Taiwan
Ming-Syan Chen, National Taiwan University, Taiwan
Qiming Chen, HP Laboratories, USA
David Cheung, The University of Hong Kong, China
Sunil Choenni, National Aerospace Laboratory, The Netherlands
Vic Ciesielski, RMIT University, Australia
W. Bruce Croft, University of Massachusetts, USA
Walter Daelemans, University of Antwerp, Belgium
Vasant Dhar, New York University, USA
Luc De Raedt, Albert-Ludwigs-University of Freiburg, Germany
Saso Dzeroski, J. Stefan Institute, Slovenia
Gillian Dobbie, Victoria University of Wellington, New Zealand
Guozhu Dong, University of South Australia, Australia
Wolfgang Essmayr, Software Competence Center Hagenberg, Austria
Martin Ester, Ludwig-Maximilians-University of Munich, Germany
Vladimir Estivill-Castro, The University of Newcastle, Australia
Wenfei Fan, Temple University, USA
Li Min Fu, University of Florida, USA
Jean-Gabriel Ganascia, Université Paris VI, France
Minos Garofalakis, Bell Laboratories, USA
Ananth Grama, Purdue University, USA

External Reviewers

Vasudha Bhatnagar, Motilal Nehru College, University of Delhi, India
Robert Bruckner, Vienna University of Technology, Austria
Dickson Chiu, Hong Kong University of Science and Technology, China
Erwin van Geenen, Nyenrode University, The Netherlands
Floris Geerts, University of Limburg, Belgium
Bart Goethals, University of Limburg, Belgium
Vivekanand Gopalkrishnan, City University of Hong Kong, China
Thanh Huynh, Vienna University of Technology, Austria
Jia-Lien Hsu, National Tsing Hua University, Taiwan
Torgeir R. Hvidsten, Norwegian University of Science and Technology, Norway
Panos Kalnis, Hong Kong University of Science and Technology, China
Juha Kärkkäinen, Max-Planck-Institut für Informatik, Germany
Mong-Li Lee, National University of Singapore, Singapore
Herman Midelfart, Norwegian University of Science and Technology, Norway
Binh Nguyen, Vienna University of Technology, Austria
Trong Dung Nguyen, Japan Advanced Institute of Science and Technology, Japan
Jian Pei, Simon Fraser University, Canada
Andreas Rauber, Vienna University of Technology, Austria
Graeme Richards,University of East Anglia, United Kingdom
Francesco Scarcello, University of Calabria, Italy
Domenico Talia, ISI-CNR, Italy
Thomas Thalhammer, Johannes Kepler University of Linz, Austria
Ljupčo Todorovski, J. Stefan Institute, Slovenia
Martin Wehr, stb software technology beratung ag, Germany
Alicja Wieczorkowska, University of North Carolina Charlotte, USA
Yi-Hung Wu, National Tsing Hua University, Taiwan

Table of Contents

Knowledge Management in Heterogeneous Data Warehouse Environments

Larry Kerschberg

Co-Director, E-Center for E-Business,
Department of Information and Software Engineering, George Mason University,
MSN 4A4, 4400 University Drive, Fairfax, VA 22030-4444, USA
kersch@gmu.edu; http://eceb.gmu.edu/

Abstract. This paper addresses issues related to Knowledge Management in the context of heterogeneous data warehouse environments. The traditional notion of data warehouse is evolving into a federated warehouse augmented by a knowledge repository, together with a set of processes and services to support enterprise knowledge creation, refinement, indexing, dissemination and evolution.

1 Introduction

Today's organizations are creating data and information at an unprecedented pace. Much of that data comes from organizational business transactions. Traditionally, that data has resided on corporate servers and has represented operational on-line transaction processing (OLTP) data. The goal of a *data warehouse* is to integrate applications at the *data level*. The data is extracted from operational systems, cleansed, transformed, and placed into the data warehouse or data mart according to a schema, such as the star or snowflake schema [1]. Although the notion of creating an *integrated* data warehouse is appealing conceptually, it may be infeasible operationally. Trends indicate that *federated* data warehouse architectures are more practical, from the political, operational, and technical points-of-view [2, 3].

Moreover, as organizations move their operations to the Internet and establish partnerships, via portals and extranets, with both their customers and suppliers, the "data" for the *e-enterprise* becomes distributed among many parties. This presents both challenges and opportunities in building enhanced data warehouses that reflect the information holdings of the e-enterprise.

Our notion of the data warehouse must be extended to include not only operational transaction-oriented data, but also data created by *knowledge workers* within the enterprise. We can then include technical reports, correspondences, presentations, audio, video, maps and other *heterogeneous* data types, as well as unstructured data.

Increasingly, we view the Internet and the World Wide Web [4] as data sources that complement e-enterprise information holdings. The data collected from the Web must be incorporated into the data warehouse for business decision-making.

Data warehouse holdings can be used for *business intelligence*, based upon knowledge created by means of data mining and knowledge discovery, which are major themes of this conference. However, in order to acquire, catalog, organize, index, store and distribute the enterprise's information holdings, we need to address issues related to the management of knowledge resources, termed *knowledge management*. In this paper, we present an architecture for the management of enterprise knowledge assets in the context of data warehouses.

The paper is organized as follows. Section 2 presents an analysis of the evolution data warehouse architectures in response to evolving enterprise requirements. Section 3 presents an architecture for knowledge management, and discusses the knowledge-oriented services provided to support heterogeneous data warehouses. Section 4 presents our conclusions.

2 Data Warehouse Architectures

Data warehouse architectures have evolved in response to our evolving data and information requirements. Initially, the data warehouse was used to extract transactional data from operational systems to perform on-line analytical processing (OLAP) [5]. One of the problems with that centralized approach is that data in the warehouse is not synchronized with data residing in the underlying data sources. This has led to research on the view materialization problem [6, 7].

Although the goal of data warehousing had been to create a *centralized and unified view* of enterprise data holdings, this goal has not been fully realized. Many factors contributed to this, such a problem of semantic heterogeneity, terminology conflicts, etc. However, one of the overriding factors has been the need for organizations to assert ownership over the data. Organizations wish to *own* their data, wish to assume responsibility for the *curation*, or *stewardship* of their data, and wish to *share* their data according to well-defined sharing agreements. Thus, rather than spend large amounts of money on a centralized data warehouse, enterprises are implementing smaller data marts, and integrating them through federated architectures.

2.1 Federated Data Warehouses

The trend away from the centralized data warehouse leads to the notion of a *federated* data warehouse, whereby independent smaller warehouses within the corporation publish selected data in *data marts*, which are then integrated. One interesting case study is that of Prudential Insurance Company [8], which decided to build a federated data warehouse by combining data from a central meta-data repository with existing stand-alone Line-of-Business (LOB) warehouses. This approach turned out to be easier and less time- and resource-consuming. Before the federated warehouse, marketing personnel could not determine how many customers had both Prudential life and property insurance, in order to sell them securities. The data resided in distinct LOB data warehouses, implemented using differing commercial products and query languages. However, such queries are common in business decision-making, and the inability to answer such questions leads to lost revenues.

The Prudential approach involves the creation of an information hub containing data of interest to the entire corporation, while data of interest to LOBs remains in the local data warehouses. The hub can them be used to create specialized data marts for decision-makers.

The federated approach is appealing in that it common data can reside in a metadata repository, while rapidly changing data can reside in data marts or in the underlying data sources. We have had extensive experience with the federated approach [2, 9-11] and these concepts were proposed for the NASA EOSDIS Independent Architecture Study and they are now being implemented. We now turn our attention to another phenomenon that is imposing new requirements on data warehouses, that of e-business.

2.2 E-Enterprise Warehouses

The e-enterprise is based on inter-enterprise partnerships among customers and suppliers. These partnerships are predicated on the sharing of data, information and knowledge through interoperable business processes, data sharing protocols, and open standards such as XML [12].

E-enterprise strategic partnerships entail data that currently is distributed among specialized software-vendor-specific applications for Customer Relationship Management (CRM), Content Management (CM) for catalog integration and data meta-tagging, Enterprise Application Integration (EAI), Human Resources (HR), and back-end fulfillment systems such as Enterprise Resource Planning (ERP).

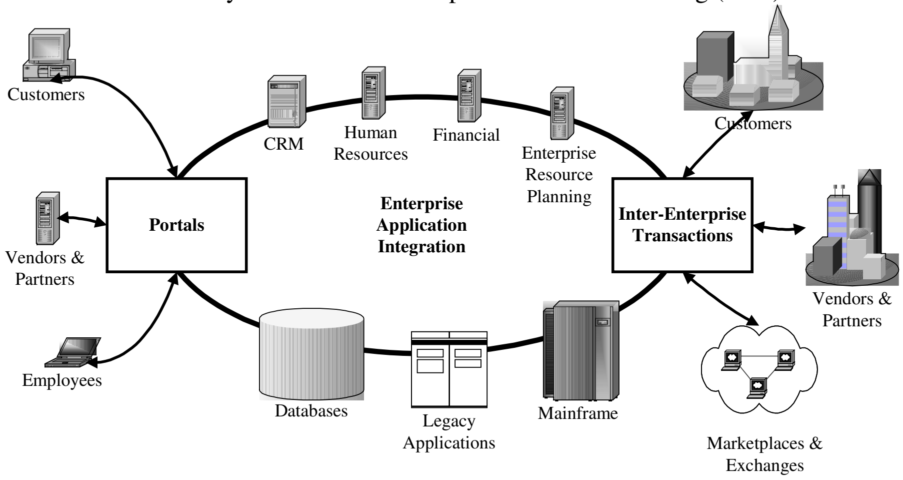


Fig. 1. E-Enterprise Heterogeneous Distributed Data Sources

The dynamic nature of the inter-enterprise relationships and the distributed heterogeneous data contained in proprietary software systems requires new approaches to data warehousing. The federated approach allows each application system to manage its data while sharing portions of its data with the warehouse. There will have to be specific agreements with customers, vendors and partners on how much of their data will be made available to the warehouse, the protocols and standards to be used to access and share data, the data security attributes and distribution permissions of shared data, and data quality standards [13, 14] to which these partners must adhere.

Another driver is the advent of the Semantic Web [15] with its associated web services [16], which will enable the creation of dynamically configurable e-enterprises. Data warehouse concepts and tools should evolve to include mechanisms to access the databases of these web services so as to obtain e-enterprise-specific data regarding those services, e.g., performance metrics, status information, meta-data, etc., which would then be stored in the federated data warehouse. We envision intelligent agents [17, 18] interacting with web service providers to gather relevant information for the data warehouse.

The data warehouse will thus become a *knowledge repository* consisting of the data from traditional data warehouses, domain knowledge relevant to enterprise decision making, workflow patterns, enterprise metadata, and enterprise ontologies.

Associated with the knowledge repository will be process knowledge for data cleansing, data quality assurance, semantic meta-tagging of data, security tagging, data stewardship (curation) and knowledge evolution. These processes are described in more detail in the next section.

3 Knowledge Management Architecture for Heterogeneous Data Warehouse Environments

The evolution of data warehouses into knowledge repositories requires a knowledge management architecture within which to acquire data from heterogeneous information sources and services, to prepare data for

The knowledge management architecture we propose supports the following:

- Access to both internal and external information sources;
- Repositories that contain explicit knowledge;
- Processes and tool support to acquire, refine, index, store, retrieve, disseminate and present knowledge;
- Mechanisms for cooperative knowledge sharing among knowledge workers;
- Organizational structures and incentives to enable and foster a knowledge sharing and learning organization;
- People who play knowledge roles within the organization, including knowledge facilitators, knowledge curators, and knowledge engineers; as well as
- Information technology support for the entire architecture.

Figure 2 below shows an architecture denoting the Knowledge Presentation, Knowledge Management, and Data Sources Layers. At the top layer, knowledge workers may communicate, collaborate and share knowledge. They are provided information by means of the Knowledge Portal, which can be tailored to the profile of each knowledge worker.

The Knowledge Management Layer depicts the Knowledge Repository and the processes that are used to acquire, refine, store, retrieve, distribute and present knowledge. These processes are used to create knowledge for the repository.

The Data Sources Layer consists of the organization's internal data sources including documents, electronic messages, web site repository, media repository of video, audio and imagery, and the domain repository consisting of the domain model, ontology, etc. Also depicted are the external sources of data, including web services that can be used to augment the internal holdings.

3.1 The Knowledge Management Process Model

The process model associated with knowledge management consists of well-defined activities which: 1) help to ensure the quality of the data and information used by knowledge workers, 2) assist in the refinement of data and information into knowledge, 3) allow the efficient storage and retrieval of metadata and knowledge, 4) promote the timely dissemination and distribution of knowledge, and 5) support the tailored presentation of knowledge. These activities are presented in the following subsections, and we review some of the major sub-activities.

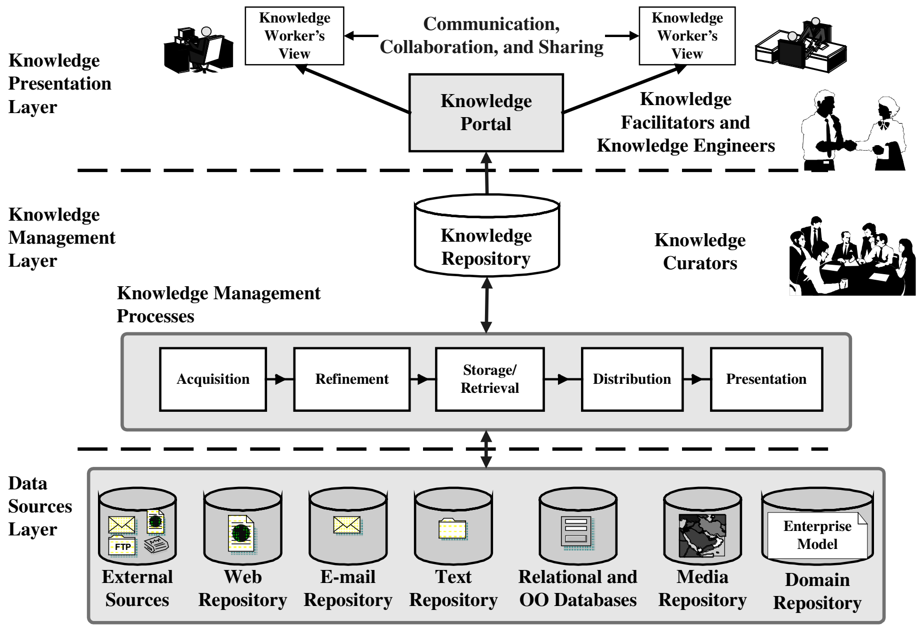


Fig. 2. Three-layer Knowledge Management Architecture

Knowledge Acquisition

During knowledge acquisition the Knowledge Engineering captures knowledge from domain experts through interviews, case histories, and other techniques. This knowledge can be represented as rules and heuristics for expert systems, or as cases for use in a case-based reasoning system.

Knowledge Refinement

Another important source of knowledge may be found in corporate repositories such as document databases, formatted transaction data, electronic messages, etc. Part of the knowledge acquisition process is to cross reference items of interest from information contained in multiple repositories under multiple heterogeneous representations. During knowledge refinement this information is classified and indexed, and metadata is created in terms of domain concepts, relationships and events. In addition, the domain context and domain usage constraints are specified. Data pedigree information is also added to the metadata descriptors, for example, intellectual property rights, data quality, source reliability, etc. Data mining and data analysis techniques can be applied to discover patterns in the data, to detect outliers, and to evolve the metadata associated with object descriptors.

Storage and Retrieval
The refined data, metadata and knowledge are indexed and stored for fast retrieval using multiple criteria, for example, by concept, by keyword, by author, by event type, and by location. In the case where discussion groups are supported, these should be indexed by thread and additional summary knowledge may be added and annotated. Access controls and security policies should be put in place to protect the knowledge base and the intellectual property it contains.

Distribution
Knowledge can be distributed in many ways, as for example, a corporate knowledge portal. Here knowledge workers may find relevant information sources for their tasks. Electronic messaging may also be used to distribute knowledge in the form of attachments of documents, presentations, etc. Another approach is to have active subscription services whereby agents inform users of relevant information in e-mail messages with hyperlinks to knowledge in the repository.

Presentation
The Knowledge Portal may handle knowledge presentation, and the interface may be tailored to the needs and preferences of each individual user. The portal should support user collaboration so as to combine tacit knowledge with explicit knowledge for problem solving.

3.2 Knowledge Management System Architecture

Figure 3 shows a conceptual model of a Knowledge Management System to support the KM processes and services. A selected number of services, which may be internal to the enterprise or outsourced through partnerships, subscriptions or web services, are now discussed.

The Knowledge Presentation and Creation Layer
The services provided at this layer enable knowledge workers to obtain personalized information via portals, to perform specialized search for information, to collaborate in the creation of new knowledge, and to transform *tacit knowledge* into *explicit knowledge* [19] via discussion groups. Our work on WebSifter [20, 21] indicates that personalized search preferences together with user-specified, ontology-directed search specification and results evaluation can enhance the precision of documents returned

by search engines. Thus search services are an important component of knowledge management.

The knowledge creation services allow knowledge workers to create value added knowledge by annotating existing knowledge, providing metatags, and aggregating heterogeneous documents into named collections for future use.

Knowledge Management Layer

This layer provides middleware services associated with knowledge indexing and information integration services (IIS). Data warehouse services are listed among the IIS, together with federation, agent, security and mediation services. Services such as data mining, metadata tagging, ontology & taxonomy, curation and workflow services are used to support the KM Process Model described in Section 3.1.

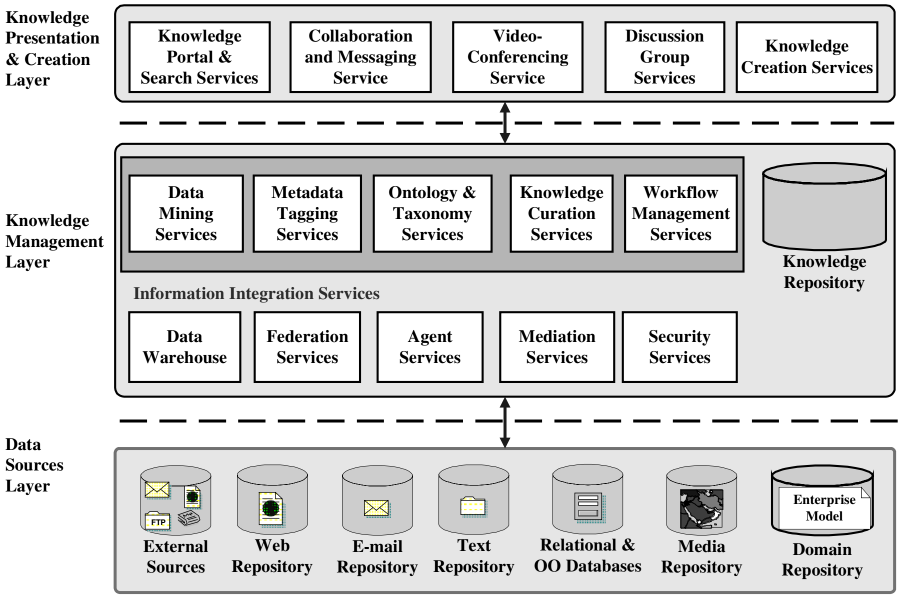

Knowledge Presentation & Creation Layer
Knowledge Portal & Search Services
Collaboration and Messaging Service
Video-Conferencing Service
Discussion Group Services
Knowledge Creation Services
Knowledge Management Layer
Data Mining Services
Metadata Tagging Services
Ontology & Taxonomy Services
Knowledge Curation Services
Workflow Management Services
Knowledge Repository
Information Integration Services
Data Warehouse
Federation Services
Agent Services
Mediation Services
Security Services
Data Sources Layer
FTP
External Sources
Web Repository
E-mail Repository
Text Repository
Relational & OO Databases
Media Repository
Enterprise Model
Domain Repository

Fig. 3. Conceptual Model for a Knowledge Management System

Data Mining Services. These services include vendor tools for deducing rules from numeric data, as well as concept mining from text. This knowledge can be used to enhance the Knowledge Repository and to provide refined knowledge to decision-makers.

Metatagging Services. Appropriate indexing of knowledge assets is crucial as collections grow. XML [12] and Resource Description Framework (RDF) [22] are emerging as open standards for tagging and metadata descriptions. The Digital Library community has proposed the Dublin Core Metadata Initiative for tagging books.

Ontology & Taxonomy Services. The organization of enterprise knowledge is an area of ongoing research. Clearly, the construction of domain-specific ontologies is of utmost importance to providing consistent and reliable terminology across the enterprise. Hierarchical taxonomies are an important classification tool. Our research in this area includes the Intelligent Thesaurus [2, 10] and we have used user-specified taxonomies to guide the WebSifter meta-search engine [20, 21]. The intelligent thesaurus is an active data/knowledge dictionary capable of supporting multiple ontologies to allow users to formulate and reformulate requests for information. The intelligent thesaurus is similar to the thesaurus found in a library; it assists analysts in identifying similar, broader or narrower terms related to a particular term, thereby increasing the likelihood of obtaining the desired information from the information sources. In addition, active rules and heuristics may be associated with object types as well as their attributes and functions. Content management in inter-enterprise environments can make use of ontologies, particularly in the area of catalog integration [23, 24].

Agent Services. As the data warehouse evolves into a knowledge repository and as the e-enterprise forms strategic partnerships with customers, partners and suppliers, we envision intelligent agents assisting in KM tasks such as: 1) monitoring e-enterprise business processes for performance information, 2) consulting authoritative external ontologies to obtain proper terminology for metatagging, 3) collecting meta-data regarding objects that flow through enterprise processes, and 4) communicating with inter-enterprise processes and partners to coordinate information interoperability.

Mediation Services. Mediation refers to a broad class of services associated with the Intelligent Integration of Information (I*3) [25]. Mediation services facilitate the extraction, matching, and integration of data from heterogeneous multi-media data sources such as maps, books, presentations, discussion threads, news reports, e-mail, etc.

One example is the mediation of temporal data of differing granularity. This is of particular importance in the context of multidimensional databases and data warehousing applications, where historical data is integrated and analyzed for patterns and interesting properties. A *temporal mediator* [26] consists of three components: 1) a repository of *windowing functions* and *conversion functions*, 2) a time unit thesaurus, and 3) a query interpreter. There are two types of windowing functions: the first associates time points to sets of object instances, and the other associates object instances to sets of time points. A conversion function transforms information in terms of one time unit into terms of some other time unit. The time unit thesaurus stores the knowledge about time units (e.g., names of time units and relationships among them). The time-unit thesaurus stores concepts such as the seasons, fiscal year definitions, and calendars, and translates these time units into others.

Users pose queries using the windowing functions and desired time units using a temporal relational algebra. To answer such a user query, the query interpreter first employs the windowing functions together with the time unit thesaurus to access the temporal data from the underlying databases and then uses the time unit thesaurus to select suitable conversion functions, which convert the responses to the desired time units.

4 Conclusions

Knowledge-based support for decision-making is becoming a key element of an enterprise's business strategy. Traditional data warehouses will evolve into knowledge management environments that handle not only operational transaction-oriented data, but also semi-structured, heterogeneous information culled from external sources and integrated into decision-oriented knowledge for enterprise decision-makers. This paper proposes a knowledge management process model together with a collection of services that can be used to manage and encourage knowledge creation within the enterprise. The key to the successful implementation of enterprise knowledge management is top management support for the creation of a knowledge sharing and learning organization.

Acknowledgements. The research was sponsored, in part, by an infrastructure grant from the Virginia Center for Information Technology, and by the E-Center for E-Business industry sponsors. The author thanks Benjamin S. Kerschberg for helpful discussions.

References

[1] R. Kimball, *The Data Warehouse Toolkit: Practical Techniques for Building Dimensional Data Warehouses*: John Wiley & Sons, Inc., 1996.

[2] L. Kerschberg and D. Weishar, "Conceptual Models and Architectures for Advanced Information Systems," *Applied Intelligence*, vol. 13, pp. 149-164, 2000.

[3] J. M. Firestone, "DKMS Brief No. Nine: Enterprise Integration, Data Federation, and DKMS: A Commentary," Executive Information Systems, 1999.

[4] T. Berners-Lee, R. Cailliau, A. Loutonen, H. F. Nielsen, and A. Secret, "The World-Wide Web," *Communications of the ACM*, vol. 37, pp. 76—82, 1994.

[5] W. H. Inmon and R. D. Hackathorn, *Using the Data Warehouse*. New York: John Wiley & Sons, Inc., 1994.

[6] R. Alonso, D. Barbara, and H. Garcia-Molina, "Data Caching Issues in an Information Retrieval System," *ACM Transactions on Database Systems*, vol. 15, 1990.

[7] L. Seligman and L. Kerschberg, "A Mediator for Approximate Consistency: Supporting 'Good Enough' Materialized Views," *Journal of Intelligent Information Systems*, vol. 8, pp. 203 - 225, 1997.

[8] J. Moad, "Extra Helping of Data," in *PC Week Online*, 1998.

[9] D. A. Menascé, H. Gomaa, and L. Kerschberg, "A Performance-Oriented Design Methodology for Large-Scale Distributed Data Intensive Information Systems," presented at First IEEE International Conference on Engineering of Complex Computer Systems (Best Paper Award), Florida, 1995.

[10] L. Kerschberg, H. Gomaa, D. A. Menascé, and J. P. Yoon, "Data and Information Architectures for Large-Scale Distributed Data Intensive Information Systems," presented at Proc. of the Eighth IEEE International Conference on Scientific and Statistical Database Management, Stockholm, Sweden, 1996.

[11] H. Gomaa, D. Menascé, and L. Kerschberg, “A Software Architectural Design Method for Large-Scale Distributed Information Systems,” *Journal of Distributed Systems Engineering*, 1996.

[12] W3C, “Extensible Markup Language (XML); http://www.w3.org/XML/,” 2001.

[13] A. Motro and I. Rakov, “Estimating the Quality of Databases,” in *Proceedings of FQAS 98 Third International Conference on Flexible Query Answering Systems, Lecture Notes in Artificial Intelligence*, vol. 1495, T. Andreasen, H. Christiansen, and H. L. Larsen, Eds. Berlin: Springer-Verlag, 1998, pp. 298-307.

[14] A. Motro and P. Smets, “Uncertainty Management in Information Systems: from Needs to Solutions.” Norwall, MA: Kluwer Academic Publishers, 1996, pp. 480.

[15] J. Hendler, “Agents and the Semantic Web,” in *IEEE Intelligent Systems*, 2001, pp. 30-37.

[16] S. A. McIlraith, T. C. Son, and H. Zeng, “Semantic Web Services,” in *IEEE Intelligent Systems*, 2001, pp. 46-53.

[17] L. Kerschberg, “Knowledge Rovers: Cooperative Intelligent Agent Support for Enterprise Information Architectures,” in *Cooperative Information Agents*, vol. 1202, *Lecture Notes in Artificial Intelligence*, P. Kandzia and M. Klusch, Eds. Berlin: Springer-Verlag, 1997, pp. 79-100.

[18] L. Kerschberg, “The Role of Intelligent Agents in Advanced Information Systems,” in *Advances in Databases*, vol. 1271, *Lecture Notes in Computer Science*, C. Small, P. Douglas, R. Johnson, P. King, and N. Martin, Eds. London: Springer-Verlag, 1997, pp. 1-22.

[19] I. Nonaka and H. Takeuchi, *The Knowledge-Creating Company: How Japanese Companies Create the Dynamics of Innovation*: Oxford University Press, 1995.

[20] A. Scime and L. Kerschberg, “WebSifter: An Ontological Web-Mining Agent for E-Business,” presented at IFIP 2.6 Working Conference on Data Semantics (DS-9), Hong Kong, China, 2001.

[21] A. Scime and L. Kerschberg, “WebSifter: An Ontology-Based Personalizable Search Agent for the Web,” presented at International Conference on Digital Libraries: Research and Practice, Kyoto Japan, 2000.

[22] W3C, “Semantic Web Activity: Resource Description Framework (RDF); http://www.w3.org/RDF/,” 2001.

[23] B. Omelayenko and D. Fensel, “An Analysis of Integration Problems of XML-Based Catalogs fo B2B Electronic Commerce,” presented at IFIP 2.6 Working Conference on Data Semantics (DS-9), Hong Kong, China, 2001.

[24] D. Fensel, *Ontologies: Silver Bullet for Knowledge Management and Electronic Commerce*. Berlin: Springer-Verlag, 2001.

[25] G. Wiederhold, “Mediators in the Architecture of Future Information Systems,” *IEEE Computer*, vol. 25, 1992.

[26] X. S. Wang, C. Bettini, A. Brodsky, and S. Jajodia, “Logical Design for Temporal Databases with Multiple Temporal Types,” *ACM Transactions on Database Systems*, 1997.

Mining Generalized Association Rules with Multiple Minimum Supports

Ming-Cheng Tseng[1] and Wen-Yang Lin[2]

[1] Institute of Information Engineering, I-Shou University,
Kaohsiung 84008, Taiwan
tmc001@ksts.seed.net.tw

[2] Department of Information Management, I-Shou University,
Kaohsiung 84008, Taiwan
wylin@isu.edu.tw

Abstract. Mining generalized association rules in the presence of the taxonomy has been recognized as an important model in data mining. Earlier work on generalized association rules confined the minimum support to be uniformly specified for all items or for items within the same taxonomy level. In this paper, we extended the scope of mining generalized association rules in the presence of taxonomy to allow any form of user-specified multiple minimum supports. We discussed the problems of using classic Apriori itemset generation and presented two algorithms, MMS_Cumulate and MMS_Stratify, for discovering the generalized frequent itemsets. Empirical evaluation showed that these two algorithms are very effective and have good linear scale-up characteristic. ...

1 Introduction

Mining association rules from a large database of business data, such as transaction records, has been a hot topic within the area of data mining. This problem is motivated by applications known as market basket analysis to find relationships between items purchased by customers, that is, what kinds of products tend to be purchased together [1].

An association rule is an expression of the form $A \Rightarrow B$, where A and B are sets of items. Such a rule reveals that transactions in the database containing items in A tend to contain items in B, and the probability, measured as the fraction of transactions containing A also containing B, is called the *confidence* of the rule. The *support* of the rule is the fraction of the transactions that contain all items both in A and B. For an association rule to hold, the support and the confidence of the rule should satisfy a user-specified minimum support called *minsup* and minimum confidence called $minconf$, respectively. The problem of mining association rules is to discover all association rules that satisfy *minsup* and $minconf$.

Most early work on associations mining was focused on deriving efficient algorithms for finding frequent itemsets [2][3][6][7]. Despite the great achievement in improving the efficiency of mining algorithms, the existing association rule model used in all of these studies incurs some problems. First, in many applications, there are taxonomies (hierarchies), explicitly or implicitly, over the items. It may be more useful to find association at different levels of the taxonomy [4][8] than only at the primitive concept level. Second, in reality, the frequencies of items are not uniform. Some items occur very frequently in the transactions while others rarely appear. A uniform minimum support assumption would hinder the discovery of some deviations or exceptions that are more interesting but much less supported than general trends. Furthermore, a single minimum support also ignores the fact that support requirement varies at different level when mining association rules in the presence of taxonomy.

These observations lead us to the investigation of mining generalized association rules across different levels of the taxonomy with non-uniform minimum supports. The methods we propose not only can discover associations that span different hierarchy levels but also have high potential to produce rare but informative item rules.

The remaining of the paper is organized as follows. A review of related work is given in Section 2. The problem of mining generalized association rules with multiple minimum supports is formalized in Section 3. In Section 4, we explain the proposed algorithms for finding frequent itemsets. The evaluation of the proposed algorithms on IBM synthetic data is described in Section 5. Finally, our conclusions are stated in Section 6.

2 Related Work

The problem of mining association rules in presence of taxonomy information is first addressed in [4] and [8], independently. In [8], the problem is named as mining generalized association rules, which aims to find associations between items at any level of the taxonomy under the *minsup* and *minconf* constraint. Their work, however, doesn't recognize the varied support requirements inherent in items at different hierarchy levels.

In [4], the problem mentioned is somewhat different from that considered in [8]. They have generalized the uniform minimum support constraint to a form of level-wise assignment, i.e., items at the same level receive the same minimum support, and aims at mining associations level-by-level in a fixed hierarchy. That is, only associations between items at the same level are examined progressively from the top level to the bottom. Though their approach also is capable of mining level-acrossing associations, the minimum supports of items are specified as uniform in each taxonomy level. This restrains the flexibility and power of association rules mining.

Another form of association rules mining with multiple minimum supports has been proposed in [5] recently. Their method allows the users to specify different minimum supports to different items and can find rules involving both frequent and rare items. However, their model considers no taxonomy at all and hence fails to find associations between items at different hierarchy levels.

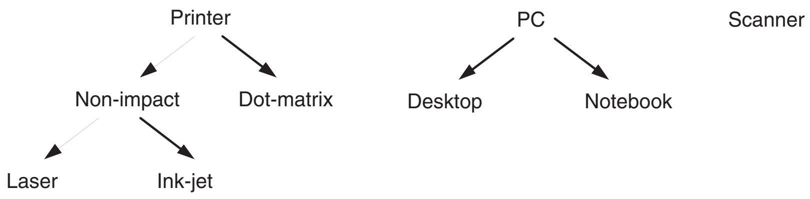


Fig. 1. An example taxonomy

3 Problem Statement

3.1 Mining Generalized Association Rules

Let $\mathcal{I} = \{i_1, i_2, \ldots, i_m\}$ be a set of items and $\mathcal{D} = \{t_1, t_2, \ldots, t_n\}$ be a set of transactions, where each transaction $t_i = \langle tid, A \rangle$ has a unique identifier tid and a set of items A ($A \subseteq \mathcal{I}$). To study the mining of generalized association rules from $\mathcal{D}$, we assume that the taxonomy of items, $\mathcal{T}$, is available and is denoted as a directed acyclic graph on $\mathcal{I} \cup \mathcal{J}$, where $\mathcal{J} = \{j_1, j_2, \ldots, j_p\}$ represents the set of generalized items derived from $\mathcal{I}$. Figure 1 illustrates a taxonomy constructed for $\mathcal{I} = \{$Laser printer, Ink-jet printer, Desktop PC, Notebook, Scanner$\}$ and $\mathcal{J} = \{$Non-impact printer, Dot matrix printer, Printer, Personal computer$\}$.

Given a transaction $t = \langle tid, A \rangle$, we say an itemset B is in t if every item in B is in A or is an ancestor of some item in A. An itemset B has support s, denoted as $s = sup(B)$, in the transaction set $\mathcal{D}$ if $s\%$ of transactions in $\mathcal{D}$ contain B. A generalized association rule is an implication of the following form $A \Rightarrow B$, where $A, B \subset \mathcal{I} \cup \mathcal{J}, A \cap B = \emptyset$, and no item in B is an ancestor of any item in A. The support of this rule, $sup(A \Rightarrow B)$, is equal to the support of $A \cup B$. The confidence of the rule, $conf(A \Rightarrow B)$, is the ratio of $sup(A \cup B)$ versus $sup(A)$. The problem of mining generalized association rules is that, given a set of transactions $\mathcal{D}$ and a taxonomy $\mathcal{T}$, find all generalized association rules that has support and confidence greater than a user-specified minimum support ($minsup$) and minimum confidence ($minconf$), respectively.

3.2 Multiple-Support Specification

To allow the user to specify different minimum supports to different items, we have to extend the uniform support used in generalized association rules. The definition of generalized association rules remains the same but the minimum support is changed.

Let $ms(a)$ denote the minimum support of an item a in $\mathcal{I} \cup \mathcal{J}$. An itemset $A = \{a_1, a_2, ..., a_k\}$, where $a_i \in \mathcal{I} \cup \mathcal{J}$, is frequent if the support of A is larger than the lowest value of minimum support of items in A, i.e., $sup(A) \geq \min_{a_i \in A} ms(a_i)$. A generalized association rule $A \Rightarrow B$ is strong if $sup(A \Rightarrow B) \geq \min_{a_i \in A \cup B} ms(a_i)$ and $conf(A \Rightarrow B) \geq minconf$. The problem of mining generalized association

Table 1. A transaction database($\mathcal{D}$)

tid	Items Purchased
11	Notebook, Laser printer
12	Scanner, Dot-matrix printer
13	Dot-matrix printer, Ink-jet printer
14	Notebook, Dot-matrix printer, Laser printer
15	Scanner
16	Desktop computer

rules with multiple minimum item supports $ms(a_1), ms(a_2), ..., ms(a_n)$ is to find all generalized association rules that are strong.

Example 1. Consider a shopping transaction database $\mathcal{D}$ in Table 1 and the taxonomy $\mathcal{T}$ as shown in Figure 1. Let the minimum support associated with each item in the taxonomy be as below:

$ms(\textsf{Printer}) = 80\%$	$ms(\textsf{Non-impact}) = 65\%$	$ms(\textsf{Dot matrix}) = 70\%$
$ms(\textsf{Laser}) = 25\%$	$ms(\textsf{Ink-jet}) = 60\%$	$ms(\textsf{Scanner}) = 15\%$
$ms(\textsf{PC}) = 35\%$	$ms(\textsf{Desktop}) = 25\%$	$ms(\textsf{Notebook}) = 25\%$

Let *minconf* be 60%. The support of the following generalized association rule

$$\textsf{PC}, \textsf{Laser} \Rightarrow \textsf{Dot-matrix}\ (sup = 16.7\%, conf = 50\%)$$

is less than $\min\{ms(\textsf{PC}), ms(\textsf{Laser}), ms(\textsf{Dot-matrix})\} = 25\%$, thus makes this rule fail. But another rule

$$\textsf{PC} \Rightarrow \textsf{Laser}(sup = 33.3\%, conf = 66.7\%)$$

holds because the support satisfies $\min\{ms(\textsf{PC}), ms(\textsf{Laser})\} = 25\%$, and the confidence is larger than $minconf$, respectively.

4 Methods for Generating Frequent Itemsets

4.1 Algorithm Basics

Intuitively, the process of mining generalized association rules with multiple minimum supports is the same as that used in traditional association rules mining: First discovering all frequent itemsets, and then from these itemsets generating rules that have large confidence. Since the second phase is straightforward, we concentrate on algorithms for finding all frequent itemsets. We propose two methods, called MMS_Cumulate and MMS_Stratify, which generalize of the Cumulate and Stratify algorithms presented in [8]; MMS stands for Multiple Minimum Supports.

Let k-itemset denote an itemset with k items. Our algorithm follows the level-wise approach widely used in most efficient algorithms to generate all frequent k-itemsets. First, scan the whole database $\mathcal{D}$ and count the occurrence of each item to generate the set of all frequent 1-itemsets (L_1). In each subsequent step $k, k \geq 2$, the set of frequent k-itemsets, L_k, is generated as follows: 1) Generate a set of candidate k-itemsets, C_k, from L_{k-1}, using the apriori-gen procedure described in [2]; 2) Scan the database $\mathcal{D}$, count the occurrence of each itemset in C_k, and prune those with less support.

The effectiveness of such approach heavily relies on a *downward closure property*: if any subset of a k-itemset is not frequent, then neither is the k-itemset. Hence, we can preprune some less supported k-itemsets in the course of examining $(k-1)$-itemsets. This property, however, may fail in the case of multiple minimum supports. For example, consider four items a, b, c, and d that have minimum supports specified as $ms(a) = 15\%$, $ms(b) = 20\%$, $ms(c) = 4\%$, and $ms(d) = 6\%$. Clearly, a 2-itemset $\{a, b\}$ with 10% of support is discarded for $10\% < \min\{ms(a), ms(b)\}$. According to the downward closure, the 3-itemsets $\{a, b, c\}$ and $\{a, b, d\}$ will be pruned even though their supports may be larger than $ms(c)$ and $ms(d)$, respectively.

To solve this problem, Liu, et al. [5] have proposed a concept called *sorted closure property*, which assume that all items within an itemset are sorted in increasing order of their minimum supports. Since [5] did not clarify this important property, we hence give a formalization. Hereafter, to distinguish from the traditional itemset, a sorted k-itemset is denoted as $\langle a_1, a_2, \ldots, a_k \rangle$.

Lemma 1 (Sorted closure). *If a sorted k-itemset $\langle a_1, a_2, \ldots, a_k \rangle$, for $k \geq 2$ and $ms(a_1) \leq ms(a_2) \leq \ldots \leq ms(a_k)$, is frequent, then all of its sorted subsets with $k-1$ items are frequent, except the subset $\langle a_2, a_3, \ldots, a_k \rangle$.*

The result in Lemma 1 reveals the obstacle in using the apriori-gen in generating frequent itemsets. For example, consider a sorted candidate 2-itemset $\langle a, b \rangle$. It is easy to find if we want to generate this itemset from L_1, both items a and b should be included in L_1; that is, each one should be occurring more frequently than the corresponding minimum support $ms(a)$ and $ms(b)$. Clearly, the case $ms(a) \leq sup(a) \leq sup(b) < ms(b)$ fails to generate $\langle a, b \rangle$ in C_2 even $sup(\langle a, b \rangle) \geq ms(a)$.

To solve this problem, [5] suggest using a sorted itemset, called *frontier set*, $F = (a_j, a_{j_1}, a_{j_2}, \ldots, a_{j_l})$, to generate the set of candidate 2-itemsets, where $a_j = \min_{a_i \in \mathcal{I} \cup \mathcal{J}} \{a_i | sup(a_i) \geq ms(a_i)\}$, $ms(a_j) \leq ms(a_{j_1}) \leq ms(a_{j_2}) \leq \ldots \leq ms(a_{j_l})$, and $sup(a_{j_i}) \geq ms(a_j)$, for $1 \leq i \leq l$. For details on generating C_2 and C_k, $k \geq 3$ using F, please see [5].

4.2 Algorithm MMS_Cumulate

As stated in [8], the main problem arisen from incorporating taxonomy information into association rules mining is how to effectively computing the occurrences of an itemset A in the transaction database $\mathcal{D}$. This involves checking for each item $a \in A$ whether a or any of its descendants are contained in a transaction t. Intuitively, we can simplify the task by first adding the ancestors of all items in a transaction t into t. Then a transaction t contains A if and only if the extended transaction t^+ contains A. Our MMS_Cumulate is deployed based on this simple concept with enhancements:

Lemma 2. [8] *The support of an itemset A that contains both an item a and its ancestor $\hat{a}$ is the same as the support for itemset $A - \{a\}$.*

Lemma 3. *An item a that is not present in any itemset of L_k will not be present in any itemset of C_{k+1}.*

Enhancement 1. *Ancestor filtering*: only ancestors that are in one or more candidates of the current C_k are added into a transaction. That is, any ancestor that is not present in any of the candidates in C_k is pruned.

Enhancement 2. *Itemset pruning*: in each C_k, any itemset that contains both an item and its ancestor is pruned. This is derived from Lemma 2. But note that the pruning should be performed for each C_k ($k \geq 2$), instead of C_2 only[1].

Enhancement 3. *Item-pruning*: an item in a transaction t can be pruned if it is not present in any of the candidates in C_k, as justified by Lemma 3.

Indeed, the concept of Enhancement 2 is derived from Lemma 3 as well. Since an item may be a terminal or an interior node in the taxonomy graph and the transactions in database $\mathcal{D}$ are composed of terminal items only, we have to perform ancestor-filtering first and then item-pruning; otherwise, we will lose the case that though some items are not frequent, in contrast are their ancestors. Figure 2 shows an overview of the MMS_Cumulate algorithm.

4.3 Algorithm MMS_Stratify

The concept of stratification is introduced in [8]. It refers to a level-wise counting strategy from the top level of the taxonomy down to the lowest level, hoping that candidates containing items at higher levels will not have minimum support, yielding no need to count candidates which include items at lower levels. However, this counting strategy may fail in the case of non-uniform minimum supports.

Example 2. Let {Printer, PC}, {Printer, Desktop}, and {Printer, Notebook} are candidate itemsets to be count. The taxonomy and minimum supports are defined as Example 1. Using the level-wise strategy, we first count {Printer, PC} and

[1] The statement of Lemma 2 in [8] is incorrect. For example, consider two itemsets $\{a, b\}$ and $\{a, c\}$ in L_2, and c is an ancestor of b. Note that b and c are not in the same itemset, but clearly $\{a, b, c\}$ will be in C_3. This implies that we have to perform the pruning not only in C_2 but also all subsequent C_k, for $k \geq 3$.

```
Create IMS; /* the table of user-defined minimum support */
Create IA; /* the table of each item and its ancestors from taxonomy $\mathcal{T}$ */
SMS = sort(IMS); /* ascending sort according to $ms(a)$ stored in IMS */
$F$ = $F$-gen(SMS, $\mathcal{D}$, IA); /* see [5] */
$L_1 = \{a \in F | sup(a) \geq ms(a)\}$;
for ( $k = 2$; $L_{k-1} \neq \emptyset$;$k++$) do
  if $k = 2$ then $C_2 = C_2$-gen($F$); /* see [5] */
  else $C_k = C_k$-gen($L_{k-1}$); /* see [5] */
  Delete any candidate in $C_k$ that consists of an item and its ancestor;
  Delete any ancestor in IA that is not present in any of the candidates in $C_k$;
  Delete any item in $F$ that is not present in any of the candidates in $C_k$;
  for each transaction $t \in \mathcal{D}$ do
    for each item $a \in t$ do
      Add all ancestors of $a$ in IA into $t$;
    Remove any duplicate from $t$;
    Delete any item in $t$ that is not present in $F$;
    $C_t$ = subset($C_k$, $t$); /* see [2] */
    for each candidate $A \in C_t$ do
      Increment the count of $A$;
  end for
  $L_k = \{A \in C_k | sup(A) \geq ms(A[1])\}$; /* $A[1]$ denote the first item in $A$ */
end for
Result = $\bigcup_k L_k$;
```

Fig. 2. Algorithm MMS_Cumulate

assume that it is not frequent, i.e., $sup(\{\mathsf{Printer}, \mathsf{PC}\}) < 0.35$. Since the minimum supports of $\{\mathsf{Printer}, \mathsf{Desktop}\}$, 0.25, and $\{\mathsf{Printer}, \mathsf{Notebook}\}$, also 0.25, are less than that of $\{\mathsf{Printer}, \mathsf{PC}\}$, we cannot assure that the occurrences of $\{\mathsf{Printer}, \mathsf{Desktop}\}$ and $\{\mathsf{Printer}, \mathsf{Notebook}\}$, though less than $\{\mathsf{Printer}, \mathsf{PC}\}$, are also less than their minimum supports. In this case, we still have to count $\{\mathsf{Printer}, \mathsf{Desktop}\}$ and $\{\mathsf{Printer}, \mathsf{Notebook}\}$ even $\{\mathsf{Printer}, \mathsf{PC}\}$ do not have minimum support.

The following observation inspires us to the deployment of the MMS_Stratify algorithm.

Lemma 4. *Consider two k-itemsets* $\langle a_1, a_2, \ldots, a_k \rangle$ *and* $\langle a_1, a_2, \ldots, \hat{a}_k \rangle$*, where* $\hat{a}_k$ *is an ancestor of* a_k*. If* $\langle a_1, a_2, \ldots, \hat{a}_k \rangle$ *is not frequent, then neither is* $\langle a_1, a_2, \ldots, a_k \rangle$*.*

Lemma 4 implies that if a sorted candidate itemset in higher level of the taxonomy is not frequent, then neither are all of its descendants that differ from the itemset only in the last item. Note that we do not make any relative assumption about the minimum supports of the item a_k and its ancestor $\hat{a}_k$. This means that the claim in Lemma 4 applies to any specifications of $ms(a_k) = ms(\hat{a}_k)$ (corresponding to uniform case), $ms(a_k) < ms(\hat{a}_k)$, or $ms(a_k) > ms(\hat{a}_k)$

(not ordinary case). As will be clear later, this makes our counting strategy applicable to any specifications of user-specified minimum item supports.

We first divide C_k, according to the ancestor-descendant relationship claimed in Lemma 4, into two disjoint subsets, called *top candidate set* TC_k and *residual candidate set* RC_k, defined as follows. Consider a set, S_k, of candidates in C_k induced by the schema $\langle a_1, a_2, \ldots, a_{k-1}, *\rangle$, where $'*'$ denotes don't care. A candidate k-itemset $A = \langle a_1, a_2, \ldots, a_{k-1}, a_k\rangle$ is a top candidate if none of the candidates in S_k is an ancestor of A. That is,

$$TC_k = \{A | A \in C_k; \forall \bar{A} \in C_k, A[i] = \bar{A}[i], 1 \le i \le k-1, A[k] = ancestor(\bar{A}[k])\},$$

and

$$RC_k = C_k - TC_k.$$

Example 3. Assume that the candidate 2-itemset C_2 for Example 1 consists of ⟨Scanner, PC⟩, ⟨Scanner, Desktop⟩, ⟨Scanner, Notebook⟩, ⟨Notebook, Laser⟩, ⟨Notebook, Non-impact⟩, and ⟨Notebook, Dot-matrix⟩, and the supports of the items in higher levels are larger than those in lower levels. Then $TC_2 =$ {⟨Scanner, PC⟩, ⟨Notebook, Non-impact⟩, ⟨Notebook, Dot matrix⟩} and $RC_2 =$ {⟨Scanner, Desktop⟩, ⟨Scanner, Notebook⟩, ⟨Notebook, Laser⟩}.

Our approach is that, in each pass k, rather than counting all candidates in C_k as in MMS_Cumulate, we count the supports of candidates in TC_k, deleting as well as their descendants in RC_k the candidates that do not have minimum support. If RC_k is not empty, we then perform an extra scanning over the transaction database $\mathcal{D}$ to count the remaining candidates in RC_k. Again, the less frequent candidates are eliminated. The resulting TC_k and RC_k, called TL_k and RL_k respectively, form the set of frequent k-itemsets L_k. The enhancements used in the MMS_Cumulate algorithm apply to this algorithm as well.

5 Experiments

In this section, we evaluate the performance of algorithms MMS_Cumulate and MMS_Stratify, using the synthetic dataset generated in [2][8]. The parameter settings are shown in Table 2. The experiments were performed on an Intel Pentium-II 350 with 64MB RAM, running on Windows 98.

We first compare the execution times of MMS_Cumulate and MMS_Stratify for different settings of uniform minimum support, ranging from 1.0% to 2.5%. The results are shown in Figure 3. At high minimum support, MMS_Cumulate and MMS_Stratify take about the same time since there were only a few rules and most of the time was spent on scanning the database. At low minimum support, MMS_Stratify performs better than MMS_Cumulate, with the gap increasing as the minimum support decreases. The reason is that MMS_Stratify can prune much more candidates to overcompensate the time spent on scanning the database.

Table 2. Parameter settings for synthetic data generation

Parameter		Default value
$\|\mathcal{D}\|$	Number of transactions	100,000
$\|t\|$	Average size of transactions	5
$\|\mathcal{I}\|$	Average size of the maximal potentially frequent itemsets	2
N	Number of items	200
R	Number of groups	30
L	Number of levels	3
f	Fanout	5

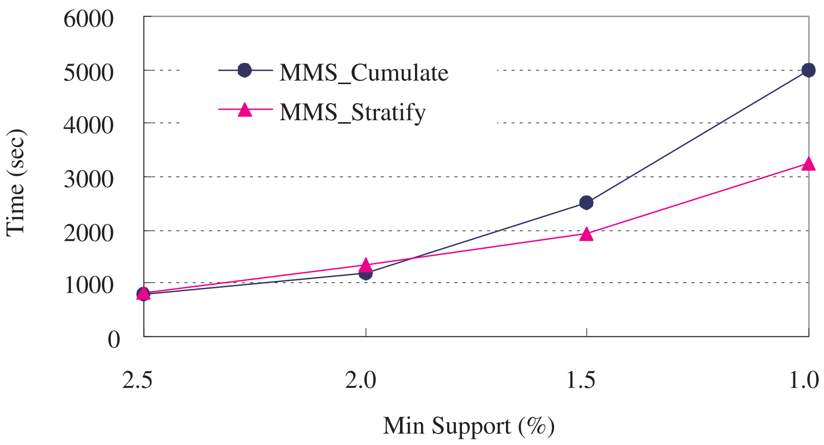


Fig. 3. Execution times for various minimum supports

We then compare the efficiency of MMS_Cumulate and MMS_Stratify for multiple minimum supports. Here, we adopt the ordinary case that the minimum support of an item a is no larger than any of its ancestors $\hat{a}$, i.e., $ms(a) \leq ms(\hat{a})$. This assumption conforms to the fact that the support of an item in the database is less than that of its ancestors. The minimum supports ranging from 1.0% to 6.0% were specified to items randomly. The results are shown in Figure 4. Algorithm MMS_Stratify performs slightly better than MMS_Cumulate, with the gap increasing as the number of transactions increases. The results also display the scalability of the algorithms. Both MMS_Cumulate and MMS_Stratify exhibit linear scale-up with the number of transactions.

6 Conclusions

We have investigated in this paper the problem of mining generalized association rules in the presence of taxonomy and multiple minimum support specification. The classic Apriori itemset generation, though works in the presence of taxonomy, fails in the case of non-uniform minimum supports. We presented two

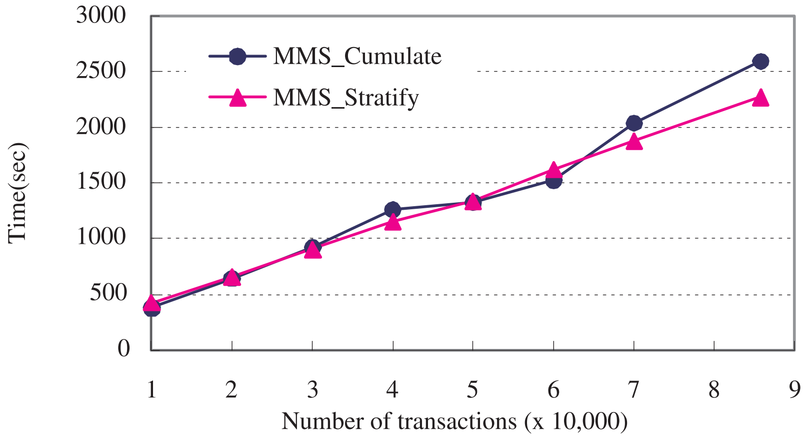


Fig. 4. Transactions scale-up for multiple min supports

algorithms, MMS_Cumulate and MMS_Stratify, for discovering these generalized frequent itemsets. Empirical evaluation showed that these two algorithms are very effective and have good linear scale-up characteristic. Between the two algorithms, MMS_Stratify performs somewhat better than MMS_Cumulate, the gap increasing with the problem size, such as the number of transactions and/or candidate itemsets.

References

1. Agrawal, R., Imielinski T., Swami, A.: Mining association rules between sets of items in large databases. Proc. 1993 ACM-SIGMOD Int. Conf. Management of Data. Washington, D.C. (1993) 207-216
2. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules. Proc. 20th Int. Conf. Very Large Data Bases. Santiago, Chile (1994) 487-499
3. Brin, S., Motwani, R., Ullman, J.D., Tsur, S.: Dynamic Itemset Counting and Implication Rules for Market-Basket Data. Proc. 1997 ACM-SIGMOD Int. Conf. Management of Data. (1997) 207-216
4. Han, J., Fu Y.: Discovery of multiple-level association rules from large databases. Proc. 21st Int. Conf. Very Large Data Bases. Zurich, Swizerland (1995) 420-431
5. Liu, B., Hsu, W., Ma, Y.: Mining association rules with multiple minimum supports. Proc. 1999 Int. Conf. Knowledge Discovery and Data Mining. (1999) 337-341
6. Park, J.S., Chen, M.S., Yu, P.S.: An effective hash-based algorithm for mining association rules. Proc. 1995 ACM-SIGMOD Int. Conf. Management of Data San Jose (1995) 175-186
7. Savasere, A., Omiecinski, E., Navathe, S.: An efficient algorithm for mining association rules in large databases. Proc. 21st Int. Conf. Very Large Data Bases. Zurich, Switzerland (1995) 432-444
8. Srikant, R., Agrawal, R.: Mining generalized association rules. Proc. 21st Int. Conf. Very Large Data Bases. Zurich, Switzerland (1995) 407-419

A Theoretical Framework for Association Mining based on the Boolean Retrieval Model*

Peter Bollmann-Sdorra[1], Alaaeldin M. Hafez[2] and Vijay V. Raghavan[2]

Abstract. Data mining has been defined as the non- trivial extraction of implicit, previously unknown and potentially useful information from data. Association mining is one of the important sub-fields in data mining, where rules that imply certain association relationships among a set of items in a transaction database are discovered.
The efforts of most researchers focus on discovering rules in the form of implications between itemsets, which are subsets of items that have adequate supports. Having itemsets as both antecedent and precedent parts was motivated by the original application pertaining to market baskets and they represent only the simplest form of predicates. This simplicity is also due in part to the lack of a theoretical framework that includes more expressive predicates.
The framework we develop derives from the observation that information retrieval and association mining are two complementary processes on the same data records or transactions. In information retrieval, given a query, we need to find the subset of records that matches the query. In contrast, in data mining, we need to find the queries (rules) having adequate number of records that support them.
In this paper we introduce the theory of association mining that is based on a model of retrieval known as the Boolean Retrieval Model. The potential implications of the proposed theory are presented.

Keywords: Data Mining, Association Mining, Theory of Association Mining, Boolean Retrieval Model.

1 Introduction

Knowledge discovery in databases is the process of identifying useful and novel structure (model) in data [7, 8, 12]. Data Mining is considered as one of the main steps in the knowledge discovery process and it is concerned with algorithms used to extract potentially valuable patterns, associations, trends, sequences and dependencies in data. Other steps in knowledge discovery include data preparation, data selection and data cleaning. Association mining is one of the central tasks in data mining [7, 14, 19]. Association mining is the process that discovers dependencies among values of certain attributes on values of some other attributes [1-4,6, 12].

* This research was supported in part by the U.S. Department of Energy, Grant No. DE-FG02-97ER1220.
[1] Fachbereich Informatik, FR 6-9, Technical University of Berlin, Franklin Str. 28/29, 10857 Berlin.
[2] ahafez(raghavan)@cacs.louisiana.edu, The Center for Advanced Computer Studies, University of Louisiana, Lafayette, LA70504-4330. USA.

Data mining techniques can discover rules that most traditional business analysis and statistical techniques fail to deliver. Furthermore, the application of data mining techniques enhances the value of data by converting expensive volumes of data into valuable rules for future tactical and strategic business development. Unfortunately, most mining techniques focus on only a narrow view of the data mining problem [7, 8, 9, 13, 15]. Researchers focus on discovering rules in the form of implications between itemsets, which are subsets of items that have adequate supports [1-4, 6, 10, 11, 14, 18, 19]. Having frequent itemsets as both antecedent and precedent parts was motivated by the original application pertaining to market baskets and they represent only the simplest form of predicates. This simplicity is also due in part to the lack of a theoretical framework that includes more expressive predicates. Unlike information retrieval systems that are supported by a strong theoretical background [5, 16, 17], where it provides advanced capabilities that give the user the power to ask more sophisticated and pertinent questions. It empowers the right people by providing the specific information they need.

The framework we develop derives from the observation that information retrieval and association mining are two complementary processes on the same data records or transactions. In information retrieval, given a query, we need to find the subset of records that matches the query. In contrast, in data mining, we need to find the queries (rules) having adequate number of records that support them.

In this paper we introduce the theory of association mining that is based on a model of retrieval known as the Boolean Retrieval Model. The theory for association mining based on this model offers a number of insights:

- a Boolean query that uses only the AND operator (i.e. a conjunction) is analogous to an itemset,
- a general Boolean query (involving AND, OR or NOT operators) has interpretation as a generalized itemset,
- notions of support of itemsets and confidence of rules can be dealt with uniformly, and
- an event algebra can be defined, involving all possible transaction subsets, to formally obtain a probability space.

In section 2, we give the problem definition. The generalized itemsets and Boolean queries are introduced in section 3. In section 4, the rules and their response strengths are given. Finally, the paper is concluded in section 5.

2. Problem Definition

2.1 Notation

• I	Set of all items $\{i_1, i_2, \ldots, i_n\}$
• 2^I	Set of all possible transactions.
• t	A transaction

• T	Set of transactions $\{t_1, t_2, \ldots, t_m\}$
• f(t)	Frequency of transaction t
• w'(t)	Weight of t
• w(t)	Normalized weight of t
• q	A query; a Boolean expression on items I
• Q*	Set of all queries q
• Q	Set of all conjunctive queries
• A	An item-set
• A_q	The item-set of all items corresponding to $q \in Q$
• RS (q)	The response set of query q
• R(q)	The response of query q
• $SS(A_q)$	Support set of item-set A_q
• $S(A_q)$	Support of item-set A_q

2.2 Association Mining

Association mining was introduced by Agrawal et al.[1], it has emerged as a prominent research area. The association mining problem also referred to as the *market basket* problem can be formally defined as follows. Let $I = \{i_1, i_2, \ldots, i_n\}$ be a set of items as $S = \{s_1, s_2, \ldots, s_m\}$ be a set of transactions, where each transaction $s_i \in S$ is a set of items that is $s_i \subseteq I$. An *association rule* denoted by $X \Rightarrow Y$, where $X, Y \subset I$ and $X \cap Y = \Phi$, describes the existence of a relationship between the two itemsets X and Y.

Several measures have been introduced to define the *strength* of the relationship between itemsets X and Y such as support, confidence, and interest. The definitions of these measures, from a probabilistic model are given below.

I. *Support* $(X \Rightarrow Y) = P(X, Y)$, or the percentage of transactions in the database that contain both X and Y.

II. *Confidence* $(X \Rightarrow Y) = P(X, Y) / P(X)$, or the percentage of transactions containing Y in transactions those contain X.

III. *Interest*($X \Rightarrow Y) = P(X, Y) / P(X)P(Y)$ represents a test of statistical independence.

2.3. Boolean Association Mining

Given a set of items $I = \{i_1, i_2, \ldots, i_n\}$, a transaction t is defined as a subset of items such that $t \in 2^I$, where $2^I = \{\emptyset, \{i_1\}, \{i_2\}, \ldots, \{i_n\}, \{i_1, i_2\}, \ldots, \{i_1, i_2, \ldots, i_n\}\}$. In reality, not all possible transactions might occur. For example, transaction $t = \emptyset$ is excluded.

Let $T \subseteq 2^I$ be a given set of transactions $\{t_1, t_2, \ldots, t_m\}$. Every transaction $t \in T$ has an assigned weight w'(t). Several possible weights could be considered,

w'(t)	=	1, for all transactions t∈T.
w'(t)	=	f(t), where f(t) is the frequency of transaction t, for all transactions t∈T, i.e., how many times the transaction t was repeated in our database.
w'(t)	=	|t| * g(t) for all transactions t∈T, where |t| is the cardinality of t, and g(t) could be either one of the weight functions w'(t)'s defined in (i) and (ii). In this case, longer transactions get higher weight.
w'(t)	=	v(t) * f(t) for all transactions t∈T, where v(t) could be the sum of the prices or profits of those items in t.

The weights w's are normalized to

$$w(t) = \frac{w'(t)}{\sum_{\forall t' \in T} w'(t')}, \text{ and } \sum_{\forall t \in T} w(t) = 1$$

Example 2.1: Let I = {beer, milk, bread} be the set of all items, where price(beer) = 5, price(milk) = 3, and price(bread) = 2. A set of transactions T along with their frequencies are given in the following table,

#	t	f(t)
1	{beer}	22
2	{milk}	8
3	{bread}	10
4	{beer, bread}	20
5	{milk, bread}	25
6	{beer, milk, bread}	15

Using weight definitions (i)-(iv), the corresponding weights are

(i)

T	{beer}	{milk}	{bread}	{beer,bread}	{milk,bread}	{beer,milk,bread}
w'(t)	1	1	1	1	1	1
w(t)	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$

(ii)

T	{beer}	{milk}	{bread}	{beer,bread}	{milk,bread}	{beer,milk, bread}
w'(t)	22	8	10	20	25	15
w(t)	0.22	0.08	0.1	0.2	0.25	0.15

(iii) using (ii) as g(t)

T	{beer}	{milk}	{bread}	{beer,bread}	{milk,bread}	{beer,milk,bread}
w'(t)	22	8	10	40	50	45
w(t)	0.13	0.05	0.06	0.23	0.27	0.26

(iv)

T	{beer}	{milk}	{bread}	{beer,bread}	{milk,bread}	{beer,milk, bread}
Price(t)	5	3	2	7	5	10
w'(t)	110	24	20	140	125	150
w(t)	0.19	0.04	0.04	0.25	0.22	0.26

3. Item-Sets and Their Representation as Queries

Item-sets describe items in transactions. In the following three definitions, we give the formal definitions for expressing item-sets as queries (i.e., logical expressions), the set of transactions making up the response set of a query, and support.

Definition 3.1: For a given set of items I, the set Q of all possible queries associated with item-sets created from I is defined as follows.
$i \in I \Rightarrow i \in Q$,
$q, q' \in Q \Rightarrow q \wedge q' \in Q$
These are all.

Definition 3.2: For any query $q \in Q$, the response set of q, RS(q), is defined as follows:
For all atomic $i \in Q$, $RS(i) = \{t \in T \mid i \in t\}$
$RS(q \wedge q') = RS(q) \cap RS(q')$

Definition 3.3: Let $q = (i_1 \wedge i_2 \wedge \ldots \wedge i_k)$ and A_q denote the item-set associated with q; that is, $A_q = \{i_1, i_2, \ldots, i_k\}$, the support of A_q is defined as

$$S(A_q) = \sum_{\forall t \in RS(q)} w(t), \text{ where } q = (i_1 \wedge i_2 \wedge \ldots \wedge i_k).$$

Lemma 3.1: The support set of A_q ; $SS(A_q)$, equals to RS(q).

Proof: Following the usual definition of a support set, the proof of lemma 3.1 is straightforward.

Example 3.1: Let w(t) be defined as in case (ii) in example 2.1. Definitions 3.1, 3.2 and 3.3 are illustrated in the following table.

	q	A_q	RS(q)	$S(A_q)$	$\frac{S(A_q)}{S(\{x,y,z\})}$
x	beer	{beer}	$\{t_1,t_4,t_6\}$	0.57	3.8
y	milk	{milk}	$\{t_2,t_5,t_6\}$	0.48	3.1
z	bread	{bread}	$\{t_3,t_4,t_5,t_6\}$	0.7	4.7
	beer∧milk	{beer,milk}	$\{t_6\}$	0.15	1.0
	beer∧bread	{beer,bread}	$\{t_4,t_6\}$	0.35	2.3
	milk∧bread	{milk,bread}	$\{t_5,t_6\}$	0.4	2.7
	beer∧milk∧bread	{beer,milk,bread}	$\{t_6\}$	0.15	1.0

Example 3.2: Let w(t) be defined as in case (iv) in example 2.1. The corresponding table is

	q	A_q	RS(q)	$S(A_q)$	$\frac{S(A_q)}{S(\{x,y,z\})}$
x	beer	{beer}	$\{t_1,t_4,t_6\}$	0.7	2.7
y	milk	{milk}	$\{t_2,t_5,t_6\}$	0.52	2.0
z	bread	{bread}	$\{t_3,t_4,t_5,t_6\}$	0.77	3.0
	beer∧milk	{beer,milk}	$\{t_6\}$	0.26	1.0
	beer∧bread	{beer,bread}	$\{t_4,t_6\}$	0.51	2.0
	milk∧bread	{milk,bread}	$\{t_5,t_6\}$	0.48	1.8
	beer∧milk∧bread	{beer,milk,bread}	$\{t_6\}$	0.26	1.0

By comparing the support values in examples 3.2 and 3.1, and because the weight values of larger transactions are considered, in example 3.2, all support values are higher than those in example 3.1. But when the relative support formula (i.e., $\frac{S(A_q)}{S(\{x,y,z\})}$) is used, all support values in example 3.1 are higher than those in example 3.2.

Lemma 3.2: For queries q, q_1, q_2 and q_3, the following axioms hold:

$RS(q \wedge q) = RS(q)$
$RS((q_1 \wedge q_2) \wedge q_3) = RS(q_1 \wedge (q_2 \wedge q_3))$
$RS(q_1 \wedge q_2) = RS(q_2 \wedge q_1)$

Example 3.3: $RS((x_1 \wedge x_2) \wedge (x_3 \wedge x_2)) = RS(x_1 \wedge x_2 \wedge x_3)$

In order to apply probabilities, we need a full set of algebra, and so, we need to redefine item-sets.

Definition 3.4: For a given set of items I, the set Q* of all possible queries is defined as follows.
$i \in I \Rightarrow i \in Q^*$,
$q, q' \in Q^* \Rightarrow q \wedge q' \in Q^*$
$q, q' \in Q^* \Rightarrow q \vee q' \in Q^*$
$q \in Q^* \Rightarrow \neg q \in Q^*$
These are all.

Definition 3.5: For any query $q \in Q^*$, the response set of transactions, R (q) is defined as

For all $i \in Q^*$, $RS(i) = \{t \in T \mid i \in t\}$
$RS(q \wedge q') = RS(q) \cap RS(q')$
$RS(q \vee q') = RS(q) \cup RS(q')$
$RS(\neg q) = T - RS(q)$

In analogy to lemma 3.2, we may apply all axioms of Boolean algebra on elements of Q*. Consequently, we can show that response sets of equivalent Boolean expressions are equal.

Theorem 3.1: If q is a transformation of q' that is obtained by applying the rules of Boolean algebra, then

$$RS(q) = RS(q')$$

Each $q \in Q^*$ can be considered as a generalized itemset. The itemsets investigated in earlier works only consider $q \in Q$.

Lemma 3.3: $\{RS(q) \mid q \in Q^*\} = 2^T$

Proof:

(i) $\{RS(q) \mid q \in Q^*\} \subseteq 2^T$ is trivial

(ii) We need to prove $2^T \subseteq \{RS(q) \mid q \in Q^*\}$.

Let $T_s = \{t_1, t_2, \ldots, t_k\} \in 2^T$, where $t_1, t_2, \ldots, t_k$ are transactions. Let $t_i = \{x_1, x_2, \ldots, x_p\}$ and $\{y_1, y_2, \ldots, y_q\} = I - T_s$. Now, if $q_i = x_1 \wedge x_2 \wedge \ldots \wedge x_p \wedge \neg y_1 \wedge \neg y_2 \wedge \ldots \wedge \neg y_q$, then $RS(q_i) = t_i$. Now, let $q = q_1 \vee q_2 \vee \ldots \vee q_k$, then $RS(q) = T_s$.

Let P be a function, where $P : 2^T \rightarrow [0,1]$. For RS(q), $q \in Q^*$, we define

$$P(RS(q)) = R(q)$$

P is well defined because,

$$RS(q_1) = RS(q_2) \Rightarrow R(q_1) = R(q_2)$$

Theorem 3.2: $(T, 2^T, P)$ is a probability space.

Proof: Check Kolmogaroff axioms.

Lemma 3.3 implies that for every subset T_s of T, such that the cardinality of T_s is greater than or equal to the minimum support, there exists at least one query $q \in Q^*$, where q is frequent. The generated queries, which are generalized itemsets, could have different complexities and lengths. In order to only generate understandable queries, new restrictions or measures, such as, compactness and simplicity, should be introduced.

4. Rules and Their Response Strengths

Let q and q' be conjunctive queries in Q.

Definition 4.1: The confidence of a rule $A_q \Rightarrow A_{q'}$ is defined as $\frac{R(q \wedge q')}{R(q)}$, (assume $A_q \cap A_{q'} = \phi$)

Definition 4.2: The interest of a rule $A_q \Rightarrow A_{q'}$ is defined as $\frac{R(q \wedge q')}{R(q) * R(q')}$, (assume $A_q \cap A_{q'} = \phi$)

Definition 4.3: The support of a rule $A_q \Rightarrow A_{q'}$ is defined as
$S(A_q \Rightarrow A_{q'}) = R(\neg q \vee q')$

Lemma 4.1: For a rule $A_q \Rightarrow A_{q'}$, $S(A_q \Rightarrow A_{q'}) = 1 - R(q) + R(q \wedge q')$

Proof:

$$\begin{aligned} SS(A_q \Rightarrow A_{q'}) &= RS(\neg q \vee q') \\ &= RS(\neg q) \cup RS(q') \\ &= (T - RS(q)) \cup RS(q') \\ &= T - (RS(q) - RS(q \wedge q')) \end{aligned}$$

$$S(A_q \Rightarrow A_{q'}) = 1 - R(q) + R(q \wedge q')$$

Lemma 4.2: For a rule $A_q \Leftrightarrow A_{q'}$,
$S(A_q \Leftrightarrow A_{q'}) = 1 - R(q) - R(q') + 2 * R(q \wedge q')$

Proof:

$$\begin{aligned} SS(A_q \Leftrightarrow A_{q'}) &= RS(\neg q \vee q') \cap RS(q \vee \neg q') \\ &= (RS(\neg q) \cup RS(q')) \cap (RS(q) \cup RS(\neg q')) \\ &= ((T - RS(q)) \cup RS(q')) \cap ((RS(q) \cup (T - RS(q')) \\ &= (T - (RS(q) \cup RS(q'))) \cup (RS(q) \cap RS(q')) \end{aligned}$$

$$S(A_q \Leftrightarrow A_{q'}) = 1 - R(q) - R(q') + 2 * R(q \wedge q')$$

For a query (or a rule) $A_q \Rightarrow A_{q'}$, the confidence measure is given in definition 4.1, and the support measure is given in definition 4.2 and lemma 4.1. Although the two measures look similar, as explained below, having the support measure instead of the confidence measure could also give information about the interest of a rule.

Let $\delta = R(q) - R(q \wedge q')$. The confidence and support of $A_q \Rightarrow A_{q'}$ are

$$conf(A_q \Rightarrow A_{q'}) = 1 - \frac{\delta}{R(q)}$$
$$S(A_q \Rightarrow A_{q'}) = 1 - \delta$$

In $conf(A_q \Rightarrow A_{q'})$, if $R(q)$ is increased and the value of δ is fixed (i.e., all new transactions are having both q and q'), the ratio $\frac{\delta}{R(q)}$ will get smaller, and

conf$(A_q \Rightarrow A_{q'})$ will get larger. If we have such a case, where all new transactions are having both q and q', then the rule $A_q \Rightarrow A_{q'}$ should not be considered as an interesting rule. Thus, in the support measure, we not only consider the confidence of a rule but also the interest of this rule.

5. Conclusions

In this paper we introduce the theory of association mining that is based on a model of retrieval known as the Boolean Retrieval Model. The framework we develop derives from the observation that information retrieval and association mining are two complementary processes on the same data records or transactions. In information retrieval, given a query, we need to find the subset of records that matches the query, while in data mining, we need to find the queries (rules) having adequate number of records that support them.

In the traditional association mining formulation, only the conjunction operator is used in forming itemsets (queries). Based on the theory of Boolean retrieval, we generalize the itemset structure by using all Boolean operators. By introducing the notion of support of generalized itemsets, a uniform measure for both itemsets and rules (generalized itemsets) has been developed. Support of a generalized itemset is extended to allow transactions to be weighted so that they can contribute to support unequally. This is achieved by the introduction of weight functions.

Since every subset of *T* has at least one generalized itemset, there can be very large number of frequent generalized itemsets, many of which could have complex structures. In order to only generate understandable queries, new restrictions or measures, such as, compactness and simplicity, should be introduced.

References

[1] R. Agrawal, T. Imilienski, and A. Swami, "Mining Association Rules between Sets of Items in Large Databases," Proc. of the ACM SIGMOD Int'l Conf. On Management of data, May 1993.

[2] R. Agrawal, and R. Srikant, "Fast Algorithms for Mining Association Rules," Proc. Of the 20^{th} VLDB Conference, Santiago, Chile, 1994.

[3] R. Agrawal, J. Shafer, "Parallel Mining of Association Rules," IEEE Transactions on Knowledge and Data Engineering, Vol. 8, No. 6, Dec. 1996.

[4] C. Agrawal, and P. Yu, "Mining Large Itemsets for Association Rules," Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 1997.

[5] A. Bookstein and W. Cooper, "A General Mathematical Model for Information Retrieval Systems," Library Quarterly, 46(2), 153-167.

[6] S. Brin, R. Motwani, J. Ullman, and S. Tsur, "Dynamic Itemset Counting and Implication Rules for Market Basket Data," SIGMOD Record (SCM Special Interset Group on Management of Data), 26,2, 1997.

[7] S. Chaudhuri, "Data Mining and Database Systems: Where is the Intersection," Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 1997.
[8] U. Fayyad, "Mining Databases: Towards Algorithms for Knowledge Discovery," Data Engineering Bulletin 21(1): 39-48 (1998)
[9] U. Fayyad: Editorial. Data Mining and Knowledge Discovery 2(1): 5-7 (1998)
[10] U. Fayyad, "Data Mining and Knowledge Discovery in Databases: Implications for Scientific Databases," SSDBM 1997: 2-11
[11] H. Mannila, "Inductive databases and condensed representations for data mining," Proc. International Logic Programming Symposium, 1997, pp. 21--30.
[12] H. Mannila, "Methods and problems in data mining," In Proceedings of the International Conference on Database Theory, Delphi, Greece, January 1997. Springer-Verlag.
[13] H. Mannila, "Data mining: machine learning, statistics, and databases," Eight International Conference on Scientific and Statistical Database Management, Stockholm. 1996.
[14] H. Mannila, H. Toivonen, and A. Verkamo, "Efficient Algorithms for Discovering Association Rules," AAAI Workshop on Knowledge Discovery in databases (KDD-94) , July 1994.
[15] A. Netz, S. Chaudhuri, J. Bernhardt and U. Fayyad, "Integration of Data Mining with Database Technology," VLDB 2000: 719-722
[16] S. E. Robertson, "Theories and Models in Information Retrieval," Journal of Documentation, 33, 126-149.
[17] G. Salton, Automatic Text Processing, Reading, MA: Addison Wesley.
[18] M. Zaki and M. Ogihara, "Theoretical foundations of association rules," In 3rd ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, June 1998.
[19] M. Zaki, S. Parthasarathy, M. Ogihara, and W. Li, " New Algorithms for Fast Discovery of Association Rules," Proc. Of the 3rd Int'l Conf. On Knowledge Discovery and data Mining (KDD-97), AAAI Press, 1997.

Mining Inter-Transactional Association Rules: Generalization and Empirical Evaluation

Ling Feng[1], Qing Li[2], and Allan Wong[3]

[1] InfoLab, Tilburg University, B 302, PO Box 90153
5000 LE Tilburg, Netherlands, ling@kub.nl
[2] Dept. of Computer Science, City University of Hong Kong
China, csqli@cs.cityu.edu.hk
[3] Dept. of Computing, Hong Kong Polytechnic University
China, csalwong@comp.polyu.edu.hk

Abstract. The problem of mining multidimensional inter-transactional association rules was recently introduced in [5, 4]. It extends the scope of mining association rules from traditional single-dimensional intra-transactional associations to multidimensional inter-transactional associations. Inter-transactional association rules can represent not only the associations of items happening within transactions as traditional intra-transactional association rules do, but also the associations of items among different transactions under a multidimensional context. "*After McDonald and Burger King open branches, KFC will open a branch two months later and one mile away*" is an example of such rules. In this paper, we extend the previous problem definition based on context expansions, and present a *generalized multidimensional inter-transactional association rule* framework. An algorithm for mining such generalized inter-transactional association rules is presented by extension of Apriori. We report our experiments on applying the algorithm to real-life data sets. Empirical evaluation shows that with the generalized inter-transactional association rules, more comprehensive and interesting association relationships can be detected.

1 Introduction

Since its introduction [1], the problem of mining association rules from large databases has been the subject of numerous studies, including efficient, Apriori-like mining methods, mining generalized, multi-level, or quantitative association rules, association rule mining query languages, constraint-based rule mining, incremental maintenance of discovered association rules, parallel and distributed mining, mining correlations and causal structures, cyclic, interesting and surprising association rule mining, and mining association rules with multiple supports.

Recently, the notion of *multidimensional inter-transactional association rules* was introduced in [5, 4]. It is motivated by the observation that many real-world associations happen under certain *contexts*. However, in the traditional association mining, this contextual information has received less exploration due to the fact that such rule mining is *intra-transaction* in nature, i.e., only looking at associations happening within the same transaction, which could be the items bought by the *same customer*, the events happening at the *same time*, etc. With the classical association rules, we can represent such knowledge as "*80% of customers who buy diapers also buy beer during the same visit*", "*When the prices of IBM and SUN go up, Microsoft's will most likely increase on the same day*". On the other hand, an inter-transactional association rule can represent not only the associations of items within transactions, but also the associations of items among different transactions along certain dimensions. "*After McDonald and Burger King open branches, KFC will open a branch two months later and one mile away*" is a 2-dimensional inter-transactional association rule example, where *time* and *space* constitute its 2-dimensional existence context. Two types of algorithms, named E/EH-Apriori (Extended/Extended Hash-based Apriori) [5, 4] and FITI (First-Intra-Then-Inter) [6], were described for mining this kind of inter-transactional association rules. A template-guided constraint-based inter-transactional association mining approach was described in [3].

However, the previous problem definition of inter-transactional association rule mining as introduced in [5, 4] has the limitation because the contexts of association relationships explored are very rigid. If we view each database transaction as a point in an m-dimensional space, each item in such an inter-transactional association rule must come from a single transaction, located at a certain contextual point, although the rule itself may embrace many items from different transactions. For ease of explanation, in the following discussions, we refer to this kind of inter-transactional association rules as *point-based inter-transactional association rules*, since only items occurring at different *contextual points* are explored with their correlationships detected.

Nevertheless, in real situations, the context under investigation is not always uniform due to the presence of possible data holes, which are analogous to the "missing parts" of a jigsaw puzzle. These data holes constitute meaningless contexts for the occurrence of any database transaction. Taking the above fast-food outlet rule for example, if one mile away from McDonald restaurant flows a wide river in some areas, then it would not be possible for any shop to be set up there. These areas thus give negative supports to the rule which in fact describes the reality that fast-food outlets usually gather together. When the mining context contains a number of holes like this, there is the risk that some rules reflecting regularities will receive unreasonably lower support/confidence compared to real situations, and some of them may be neglected by data miners.

This mining context problem, however, can be rectified by patching data holes while performing the mining. In other words, we can expand rule contexts from *point-based* (e.g., *one mile away*) to *scope-based* (e.g., *within one mile and three miles away*). In fact, for many applications, it does not matter whether an item in an inter-transactional association rule is within a single transaction or a group of transactions, provided that the contextual scope where these transactions locate is meaningful and of interest to applications. By context expansions, we can enhance the flexibility and expressiveness of inter-transactional association framework to capture more comprehensive and general knowledge like "*After McDonald and Burger King open branches, KFC will open a branch two months later and between one and three miles away*".

The aim of this paper is to extend the previous problem definition of multidimensional inter-transactional association rules given in [5, 4] based on context expansion. We call such extended association rules **generalized multidimensional inter-transactional association rules**, since they provide a *uniform* view for a number of association and sequence related patterns defined before.

The remainder of the paper is organized as follows. Section 2 gives a formal description of the problem of mining generalized multidimensional inter-transactional association rules. Section 3 describes an algorithm for mining such association rules, with the performance evaluation presented in section 4. Section 5 concludes the paper.

2 Problem Statement

An m-dimensional context can be defined through m dimensional attributes $a_1, a_2, \ldots, a_m$, whose domains $Dom(a_1)$, $Dom(a_2)$, ..., $Dom(a_m)$ are finite subsets of nonnegative integers. Such an m-dimensional space constitutes the occurrence context of association rules discussed later.

Let $n_l = (n_l.a_1, n_l.a_2, \ldots, n_l.a_m)$ and $n_u = (n_u.a_1, n_u.a_2, \ldots, n_u.a_m)$ be two *contextual points* in an m-dimensional space, whose values on the m dimensions are represented as $n_l.a_1$, $n_l.a_2$, ..., $n_l.a_m$ and $n_u.a_1$, $n_u.a_2$, ..., $n_u.a_m$, respectively. We define
1) $(n_l = n_u)$ iff $\forall a_i \in \{a_1, \ldots, a_m\}$ $(n_l.a_i = n_u.a_i)$;
2) $(n_l \preceq n_u$, conversely $n_u \succeq n_l)$ iff $\forall a_i \in \{a_1, \ldots, a_m\}$ $(n_l.a_i \leq n_u.a_i)$;
3) $(n_l \prec n_u$, conversely $n_u \succ n_l)$ iff $(n_l \preceq n_u)$ $\wedge$ $\exists a_i \in \{a_1, \ldots, a_m\}$ $(n_l.a_i < n_u.a_i)$.

Delimited by two points n_l and n_u $(n_l \preceq n_u)$, an m-dimensional *contextual scope* $[n_l,\ n_u]$ can be formed. A point n_i lies within the scope $[n_l,\ n_u]$, denoted as $n_i \in [n_l,\ n_u]$, iff $(n_l \preceq n_i \preceq n_u)$. Given two contextual scopes $[n_l,\ n_u]$ and $[n'_l,\ n'_u]$ in an m-dimensional space, the following three boolean comparison operators can be defined:
1) *inclusive*$([n_l, n_u],\ [n'_l, n'_u])$ is true, iff $(n_l \preceq n'_l) \wedge (n_u \succeq n'_u)$.
2) *intersect*$([n_l, n_u],\ [n'_l, n'_u])$ is true, iff $\exists n_i$ $(n_i \in [n_l, n_u]) \wedge (n_i \in [n'_l, n'_u])$.
3) *precedence*$([n_l, n_u],\ [n'_l, n'_u])$ is true, iff $(n_u \prec n'_l)$.

Let $\mathcal{I} = \{i_1, i_2, \ldots, i_\omega\}$ denote a set of literals, called items. A traditional transactional database $\mathcal{T}$ is a set of transactions $\{t_1, t_2, \cdots, t_n\}$, where t_i $(i = 1, 2, \ldots, n)$ is a subset of $\mathcal{I}$. Such a database model is enhanced under an m-dimensional context by associating each transaction with an m-dimensional attribute value, so that each database transaction $t \in \mathcal{T}$ can be mapped to a point n_i in an m-dimensional contextual space, describing its occurrence context.

We call a transaction $t \in \mathcal{T}$ happening at an m-dimensional contextual point n_i an *extended transaction*, and denote it as $t(n_i)$. Let $\mathcal{T}_E$ be the set of all extended transactions in the database. We call an item $i \in \mathcal{I}$ happening within

an m-dimensional contextual scope $[n_l,\ n_u]$ an *extended item*, and denote it as $i_{[n_l,\ n_u]}$. Given two extended items $i_{[n_l,\ n_u]}$ and $i'_{[n'_l,\ n'_u]}$, we define
1) $(i_{[n_l,\ n_u]} = i'_{[n'_l,\ n'_u]})$ iff $(i = i') \ \wedge \ (n_l = n'_l) \ \wedge \ (n_u = n'_u)$;
2) $(i_{[n_l,\ n_u]} < i'_{[n'_l,\ n'_u]})$ iff $(i < i') \ \vee \ (i = i' \ \wedge \ precedence([n_l,\ n_u],\ [n'_l,\ n'_u]))$.
The set of all possible extended items, $\mathcal{I}_E$, is defined as a set of $i_{[n_l,\ n_u]}$ for any $i \in \mathcal{I}$ within all possible scopes $[n_l,\ n_u]$ in the m-dimensional space.

Let $I_e = \{i_{1\ [n_{1,l},\ n_{1,u}]},\ i_{2\ [n_{2,l},\ n_{2,u}]},\ \ldots,\ i_{k\ [n_{k,l},\ n_{k,u}]}\}$ be an extended item set, where $n_{x,l} = (n_{x,l}.a_1,\ n_{x,l}.a_2,\ \ldots,\ n_{x,l}.a_m)$ and $n_{x,u} = (n_{x,u}.a_1,\ n_{x,u}.a_2,\ \ldots,\ n_{x,u}.a_m)$ $(1 \le x \le k)$. We call I_e a *normalized extended item set*, if within all the contextual scopes involved, the minimal dimensional value is 0 along each dimension, i.e., $\forall a_s \in \{a_1,\ \ldots,\ a_m\}\ Min(n_{1,l}.a_s, n_{1,u}.a_s,\ \ldots,\ n_{k,l}.a_s,\ n_{k,u}.a_s) = 0$. Let $\mathcal{I}_{NE}$ denote the set of all possible normalized extended item sets in the database.

In a similar fashion, we call a set of extended transactions, $T_e = \{t_1(n_1),\ t_2(n_2),\ \ldots,\ t_r(n_r)\}$ where $n_x = (n_x.a_1,\ n_x.a_2,\ \ldots,\ n_x.a_m)$ $(1 \le x \le r)$, a *normalized extended transaction set*, if within all the contextual points involved, the minimal dimensional value is 0 along each dimension, i.e., $\forall a_s \in \{a_1,\ \ldots,\ a_m\}$ $Min(n_1.a_s, n_2.a_s, \ldots, n_r.a_s) = 0$.

Any non-normalized extended item (transaction) set can be transformed into a normalized one through a normalization function called $Norm$, whose intention is to re-position all contextual scopes (points) in the set based on its minimal dimensional value along each dimension.

Example 1. Let $I_e = \{a_{[(0,0),\ (1,0)]},\ b_{[(1,1),\ (1,2)]}\}$, $I'_e = \{a_{[(1,0),\ (2,0)]},\ b_{[(2,1),\ (2,2)]}\}$ be two extended item sets in a 2-dimensional space. I_e is a normalized extended item set, since it has minimal value 0 for both dimensions, i.e., $min(0,1,1,1) = 0$ and $min(0,0,1,2) = 0$. But I'_e is not due to its non-zero minimal value $min(1,2,2,2) = 1$ for the first dimension. We can normalize I'_e by subtracting this minimal value 1 from the four delimiting points' first dimensional values, and then re-position the contextual scopes in I'_e as follows: $[(1\text{-}1,0),\ (2\text{-}1,0)] = [(0,0),\ (1,0)]$, $[(2\text{-}1,1),\ (2\text{-}1,2)] = [(1,1),\ (1,2)]$. The converted itemset $I''_e = \{a_{[(0,0),\ (1,0)]},\ b_{[(1,1),\ (1,2)]}\}$ turns into a normalized extended itemset.

Definition 1. *A* **generalized multidimensional inter-transactional association rule** *is an implication of the form $X \Rightarrow Y$, satisfying the following two conditions:*
1) $X \subset \mathcal{I}_E$, $Y \subset \mathcal{I}_E$, $X \cup Y \subset \mathcal{I}_{NE}$, $X \cap Y = \emptyset$;
2) $\forall i_{[n_{s,l},\ n_{s,u}]} \in (X \cup Y)$ $\nexists i_{[n'_{s,l},\ n'_{s,u}]} \in (X \cup Y)$, where $intersect([n_{s,l},\ n_{s,u}],$ $[n'_{s,l}, n'_{s,u}])$ is true.

The first clause of the definition states that only a normalized extended item set $X \cup Y$ is considered by a rule. The second clause requires that no two extended items with the same item but intersected contextual scopes co-exist in one rule. This is to avoid verbose rules like "$\underline{a_{[(0),(0)]}},\ \underline{a_{[(0),(2)]}} \Rightarrow b_{[(2),(4)]}$", "$a_{[(0),(0)]} \Rightarrow \underline{b_{[(2),(2)]}},\ \underline{b_{[(2),(4)]}}$" or "$\underline{a_{[(0),(0)]}} \Rightarrow \underline{a_{[(0),(2)]}},\ b_{[(2),(4)]}$", since the presence of $a_{[(0),(0)]}$ implies the presence of $a_{[(0),(2)]}$, and so does the pair of $b_{[(2),(2)]}$ and $b_{[(2),(4)]}$.

Based on Definition 1, a rule like "*if there is no rain within 6 hours and the weather is medium wet during the following 24 hours, then there will be no rain for 2 days*" can be expressed by a generalized 1-dimensional inter-transactional association rule "$\textit{no-rain}_{[(0),\ (1)]}$, $\textit{medium-wet}_{[(2),\ (5)]} \Rightarrow \textit{no-rain}_{[(2),\ (9)]}$". Here, each interval unit represents 6 hours.

Definition 2. *Given two subsets of a normalized extended itemset $\mathcal{I}_{ne}$, X and Y. Let T_{xy} be the set of minimal extended transaction sets that contain $X \cup Y$, and T_x be the set of minimal extended transaction sets that contain X. The* **support** *and* **confidence** *of a generalized multidimensional inter-transactional association rule $X \Rightarrow Y$ are defined as:*

$$support(X \Rightarrow Y) = |T_{xy}|/|\mathcal{T}_E|, \quad confidence(X \Rightarrow Y) = |T_{xy}|/|T_x|.$$

Given a user-specified minimum support (called *minsup*) and minimum confidence (called *minconf*), our task is to discover from a multidimensional contextual space a complete set of generalized multidimensional inter-transactional association rules with $support \geq minsup$ and $confidence \geq minconf$.

3 Mining Generalized 1-Dimensional Inter-Transactional Association Rules

Like classical association rule mining, the problem of mining generalized inter-transactional association rules can be decomposed into two subproblems: 1) large normalized extended itemset discovery; 2) generalized inter-transactional association rule derivation. Figure 1 outlines a generalized 1-dimensional inter-transactional association mining algorithm by extension of Apriori [2]. To simplify expressions, we omit bracket ( ) surrounding coordinates of points under 1-dimension, and use $[l,\ u]$ for $[(l),\ (u)]$. Also, *itemset* and *extended itemset* can be used interchangeably in the following discussion.

3.1 Pass 1

Generation of candidate set C_1. In real applications, users are usually interested in associations happening within certain limited scopes. Here, we introduce a *maxscope* threshold to specify the maximal scope of interest to applications. Given $maxscope = 3$, Figure 2 illustrates all the contextual scopes considered in our mining task. To generate candidate set C_1, for each item in $\mathcal{I}$, we attach all these possible contextual scopes, and obtain $C_1 = \{\{i_{[l,\ u]}\} \mid (i \in \mathcal{I}) \wedge (l \leq u) \wedge (0 \leq l \leq maxscope) \wedge (0 \leq u \leq maxscope)\}$. Hence, $|C_1| = |\mathcal{I}| * |\sum_{u=l}^{maxscope}(\sum_{l=0}^{maxscope} i_{[l,\ u]})| = |\mathcal{I}| * (maxscope + 1) * (maxscope + 2)/2$.

Counting candidates in C_1. As candidate 1-itemsets (e.g., $a_{[0,\ 3]}, a_{[1,\ 1]}, a_{[2,\ 3]}$) may span a set of transactions within contextual scopes (from 0 to

Input: a database DB containing an extended transaction set $\mathcal{T}_E$ under a single-dimensional context; a *minsup* threshold and a maximal contextual scope *maxscope* of interest to applications.
Output: a set of large normalized extended itemsets L discovered from the database.

```
k=1
1  $L_1 = \emptyset$;
2  $C_1 = \{\{i_{[l,\ u]}\} \mid (i \in \mathcal{I}) \ \wedge \ (l \le u) \ \wedge \ (0 \le l \le maxscope) \ \wedge (0 \le u \le maxscope)\}$;
3  foreach extended transaction $t_s(s) \in \mathcal{T}_E$ do
4      $T_s = \{t_{s+d}(s+d) \mid (t_{s+d}(s+d) \in \mathcal{T}_E) \ \wedge \ (0 \le d \le maxscope)\}$;
5      foreach extended transaction $t_{s+d}(s+d) \in T_s$ do
6          foreach item $i \in t_{s+d}$ do
7              for $(u = d;\ u \le maxscope;\ u++)$ do
8                  $i_{[d,\ u]}.count$++;    //increase along arrow $\uparrow$ shown in Figure 2
9                  for $(l = d$-1; $l \ge 0$; $l$–) do
10                     if $(i \notin t_{s+l})$ then $i_{[l,\ d]}.count$++; //increase along arrows $\nwarrow$
11                     else break;
12             endfor
13     Transform-Record-TranSet $(T_s,\ DB')$;
14 endfor
15 $L_1 = \{\{i_{[l,\ u]}\} \mid (i_{[l,\ u]} \in C_1) \ \wedge \ (i_{[l,\ u]}.count/|\mathcal{T}_E| \ge minsup)\}$

k>1
16 for $(k = 2; L_{k-1} \ne \phi; k++)$ do
17     $C_k$=E-Apriori-Gen$(L_{k-1})$;
18     foreach record $r \in DB'$ do
19         $C_r$ = E-Subset $(C_k\ , r)$;    // candidates contained in record
20         foreach candidate $X : \{i_{1[l_1,\ u_1]}, \ldots, i_{k[l_k,\ u_k]}\} \in C_r$ do   $X.count$++;
21     endfor
22     $L_k = \{X : \{i_{1[l,\ u]}, \ldots, i_{k[l_k,\ u_k]}\} \mid (X \in C_k) \ \wedge \ (X.count/|\mathcal{T}_E| \ge minsup)\}$;
23 endfor
24 $L = \bigcup_k L_k$.
```

Fig. 1. A generalized 1-dimensional inter-transactional association mining algorithm

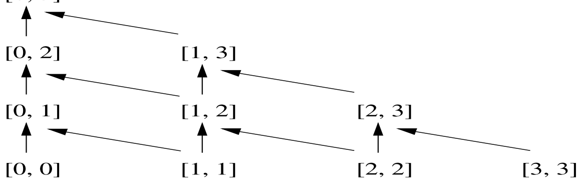


Fig. 2. 1-dimensional contextual scopes considered when *maxscope*=3

maxscope at most), to count supports of candidates, from each extended transaction $t_s(s)$ in the database, [1] we examine a set of minimal normalized extended transactions T_s instead of only itself (line 4).

Property 1. Let $T_s = \{t_{s+d}(s+d) \mid (t_{s+d}(s+d) \in \mathcal{T}_E) \wedge (0 \le d \le maxscope)\}$ be an extended transaction set. For any two extended 1-itemsets $i_{[l,\ u]}$ and $i_{[l',\ u']}$, where $(i \in \mathcal{I})$ and *inclusive* $([l', u'],\ [l, u])$ is true, if T_s contains $i_{[l,\ u]}$, then T_s also contains $i_{[l',\ u']}$.

Based on Property 1, when an item i appears in a transaction $t_{s+d}(s+d) \in T_s$ (line 6), the algorithm increases not only the counter of $i_{[d,\ d]}$, but also the counters of all those 1-itemsets $i_{[l,\ u]}$ where $inclusive([l, u],\ [d,\ d])$ along vertical and slanted arrows in Figure 2 (line 6-12). Line 10 ensures each counter increases at most by 1, given one minimal extended transaction set T_s,

Database transformation. Compared with intra-transactional association mining, from each transaction, we need to scan ($maxscope$+1) times the number of items. In order to save this extra search effort for the following passes, another work conducted during pass 1 is to transform and record every extended transaction set T_s into a new database DB'. This is performed by the function Transform-Record-TranSet $(T_s,\ DB')$ (line 13).

3.2 Pass k>1

Candidate generation. Given L_{k-1}, the candidate generation function E-Apriori-Gen(L_{k-1}) returns a superset C_k of L_k (line 17). This procedure has two parts. In the join phase, a candidate k-itemset X'' is generated from two large (k-1)-itemsets X and X', where $X = \{x_{1[l_1,u_1]}, \ldots, x_{k-2[l_{k-2},u_{k-2}]},\ x_{k-1[l_{k-1},u_{k-1}]}\}$, $X' = \{x'_{1[l'_1,u'_1]}, \ldots, x'_{k-2[l'_{k-2},u'_{k-2}]},\ x'_{k[l'_k,u'_k]}\}$, $X'' = \{x_{1[l_1,u_1]}, \ldots, x_{k-2[l_{k-2},u_{k-2}]},$ $x_{k-1[l_{k-1},u_{k-1}]},\ x'_{k[l'_k,u'_k]}\}$, [2] satisfying the following three requirements:
1) $X, X' \in L_{k-1}$; 2) $(x_{1[l_1,u_1]} = x'_{1[l'_1,u'_1]})$, $\ldots$, $(x_{k-2[l_{k-2},u_{k-2}]} = x'_{k-2[l'_{k-2},u'_{k-2}]})$, $(x_{k-1[l_{k-1},u_{k-1}]} \le x_{k[l_k,u_k]})$; 3) $X'' \in \mathcal{I}_{NE}$ (i.e., X'' is an normalized extended itemset).
When k=2, to obtain normalized candidate 2-itemsets, either $(l_1 = 0)$ in $X = \{x_{1[l_1,u_1]}\}$ or $(l'_1 = 0)$ in $X' = \{x'_{1[l'_1,u'_1]}\}$. When $k > 2$, since X, X' are normalized extended itemsets, the superset of them $X'' = X \cup X'$ is also a normalized extended itemset. Next, in the prune phase, we delete all those itemsets in C_k which have some (k-1)-subsets whose normalization forms are not in L_{k-1}.

Example 2. Let $L_2 = \{\{a_{[0,0]},\ b_{[0,1]}\},\ \{a_{[0,0]},\ c_{[1,3]}\},\ \{b_{[0,1]},\ c_{[1,3]}\},\ \{b_{[0,1]},\ d_{[2,3]}\}\}$. After the join step, $C_3 = \{\{a_{[0,0]},\ b_{[0,1]},\ c_{[1,3]}\}, \{b_{[0,1]},\ c_{[1,3]},\ d_{[2,3]}\}\}$. The prune step will delete the itemset $\{b_{[0,1]},\ c_{[1,3]},\ d_{[2,3]}\}$, because its subset $\{c_{[1,3]},\ d_{[2,3]}\}$

[1] Assume each contextual position has a transaction. We use this position as subscript to name each transaction.
[2] The extended items in an itemset are listed in an ascending order. Recall that $(i_{[n_l,\ n_u]} < i'_{[n'_l,\ n'_u]})$ iff $(i < i') \vee (i = i' \ \wedge\ precedence([n_l,\ n_u],\ [n'_l,\ n'_u]))$.

(i.e., $\{c_{[0,2]}, d_{[1,2]}\}$ after normalization) is not in L_2. We will then be left with $C_3 = \{\{a_{[0,0]}, b_{[0,1]}, c_{[1,3]}\}\}$.

Counting candidates in C_k. After generating candidate k-itemsets, the function E-Subset(C_k, r) (line 19) checks which k-itemsets in C_k are supported by a new database record r. To do this, we extract all the item IDs of these itemsets and store them in a hash tree similar to that in [2]. The contextual scopes associated with corresponding item IDs are stored uniformly in $leaf$ nodes only. Starting from the root node, we find all the candidates contained in the record r as follows. If we reach a leaf node, we first find those itemsets which have their item IDs present in r, and then further check whether the occurrence positions of these item IDs are within the specified contextual scopes indicated by the extended itemsets. If so, we add the itemsets to C_r. Considering the situation that one item ID may appear consecutively several times in one extended itemset (but with different contextual scopes of *precedence* relations, e.g., $\{a_{[0,0]}, a_{[1,2]}, b_{[1,1]}, b_{[3,3]}\}$), if we are at an interior node and have reached it by hashing the i-th item in r, we hash on each item from the i-th item **again** (rather than from the $(i{+}1)$-th item as Apriori does in [2]) and recursively apply this procedure to the node in the corresponding bucket. The subset function returns a set of k-itemsets, $C_r \subseteq C_k$, that are supported by r in the new database. We increase all the counters of k-itemsets in C_r by 1 (line 20). By scanning the transformed database DB' once, we can obtain L_k (line 22).

4 Performance Study

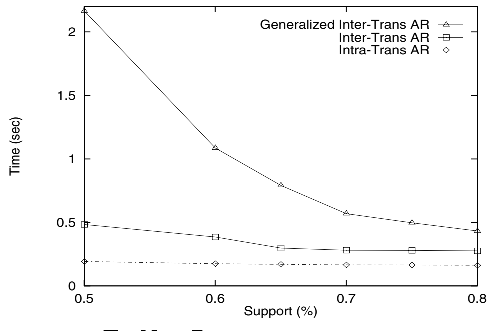
We performed two sets of experiments with meteorological data obtained from the Hong Kong Observatory headquarters, which takes meteorological observations, including *wind direction, wind speed, dry bulb temperature, relative humidity, rainfall* and *mean sea level pressure*, etc., every 6 hours each day.

Our first test is to detect generalized inter-transactional association rules from the 1996 meteorological data, and use the 1997 meteorological data from the same area in Hong Kong to examine their *predictive rate*, measured by *PredRate*$(X \Rightarrow Y) = sup(X \cup Y)/sup(X)$, i.e., the ratio of minimal extended transaction sets containing $X \cup Y$ to those containing only X. The mining context in this test is 1-dimension with *time* as its dimensional attribute. Considering seasonal changes of weather, we extract records from the first day of May to the last day of October, and there are therefore totally 736 records for each year. These raw data sets, containing continuous atmospheric elements, are further converted into appropriate formats with which the algorithm can work. 1) *Wind direction* values measured in continuous degrees are discretized into 8 wind directions - *north-east, east, south-east, south, south-west, west, north-west* and *north*; 2) *wind speed* values are classified as *light, moderate, fresh,* or *strong*; 3) *rainfall* recorded in the unit of centimeter is discretized into *no-rain, trace, light, moderate,* or *heavy*; 4) *relative humidity* is characterized into *very-dry, dry, medium-wet* or *wet*; 5) *temperature* is represented by *very-cold, cold, mild,* *warm* or *hot*; 6) *mean sea level pressure* values are discretized into *very-low, low, moderate, slightly-high, high,* or *very-high.* After transformation, we obtain 32 kinds of items in total, and each database record contains 6 different items. The interval of every two consecutive records is 6 hours.

By setting *maxscope* as 3, 7 and 11, we can detect associated meteorological relationships happening within one day ((3+1)/4=1), two days ((7+1)/4=2) and three days ((11+1)/4=3). Some generalized inter-transactional association rule examples found from the data set under $minsup = 90\%$ and $minconf = 99\%$ are as follows.

- "If there is *no rain* within 6 hours and the weather is *medium wet* during the following 24 hours, then there will be *no rain* for 2 days." ($13_{[0,1]}$, $20_{[2,5]} \Rightarrow 13_{[2,7]}$, *Pred-Rate = 96%*)
- "If it is *warm* within 2 days, then within 3 days there will be *no rain.*" ($25_{[0,7]} \Rightarrow 13_{[0,11]}$, *Pred-Rate = 76%*)
- "If the wind speed continues to be *moderate* for 2 days, then there will be *no rain* during the third day." ($10_{[0,7]} \Rightarrow 13_{[7,11]}$, *Pred-Rate = 83%*)

We compare our generalized inter-transactional association mining with the other two association mining, i.e., *traditional intra-transactional association mining* [1,2] and *point-based inter-transactional association mining* [5,4]. Table 1 summarizes the mining results. Under $minsup$=60% and $minconf$=90%, we found only 1 intra-transactional association rule and 4 point-based inter- transactional association rules, but 113 generalized inter-transactional association rules (which reduce to 27 under $minconf$=99%). This is expected as much more candidate and large itemsets are generated when mining generalized inter-transactional association mining. Note that such an increase from intra- transactional to generalized inter-transactional association rules is much greater than that from intra-transactional to point-based inter-transactional association rules, due to the relaxation of contextual scopes in detecting association relationships. Inevitably, mining generalized inter-transactional association rules demands much more time in scanning the database and counting candidates than mining the other two association rules, as illustrated in Figure 3.



T6-N32-D736, *maxscope*=3

Fig. 3. Minimum support versus execution time

Table 1. Mining Result Comparison (T6-N32-D736, *maxscope*=3)

	#Candidate-itemset	#Large-itemset	*minconf* = 90%		*minconf* = 99%	
			#Rule	Avg-Pred-Rate	#Rule	Avg-Pred-Rate
***minsup* = 60%**						
Intra-Trans. AR	33	3	1	94%	0	-
Inter-Trans. AR	332	16	4	89%	0	-
Generalized Inter-Trans. AR	801	252	113	83%	27	92%
***minsup* = 80%**						
Intra-Trans. AR	32	1	0	-	0	-
Inter-Trans. AR	323	5	1	89%	0	-
Generalized Inter-Trans. AR	379	52	27	84%	5	91%

5 Conclusion

In this paper, we have generalized the problem of mining multidimensional inter-transactional association rules based on context expansions. A more general and flexible form of association rules named **generalized multidimensional inter-transactional association rules** is presented. We described an algorithm for mining such extended association rules under 1-dimensional contexts by extension of Apriori. Empirical evaluation of the algorithm shows that with such generalized association rules, we can detect more comprehensive and interesting association relationships.

References

1. R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, pages 207–216, 1993.
2. R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. of the 20th Intl. Conf. on Very Large Data Bases*, pages 478–499, 1994.
3. L. Feng, H. Lu, J. Yu, and J. Han. Mining inter-transaction association rules with templates. In *Proc. ACM CIKM Intl. Conf. Information and Knowledge Management*, pages 225–233, 1999.
4. H. Lu, L. Feng, and J. Han. Beyond intra-transactional association analysis: Mining multi-dimensional inter-transaction association rules. *ACM Transactions on Information Systems*, 18(4):423–454, 2000.
5. H. Lu, J. Han, and L. Feng. Stock movement prediction and n-dimensional inter-transaction association rules. In *Proc. of the ACM SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery*, pages 12:1–12:7, 1998.
6. K.H. Tung, H. Lu, J. Han, and L. Feng. Breaking the barrier of transcations: Mining inter-transaction association rules. In *Proc. ACM SIGKDD Intl. Conf. Knowledge Discovery and Data Mining*, pages 297–301, 1999.

On the Equivalence of Top-down and Bottom-up Data Mining in Relational Databases

Hasan M. Jamil
Member, ACM and IEEE
jamil@acm.org

Abstract

Although knowledge discovery from large relational databases has gained popularity, and its significance is well recognized, the prohibitive nature of the cost associated with extracting such knowledge, and the lack of suitable declarative query language support, still act as limiting factors. Surprisingly, little or no relational technology has yet been significantly exploited in data mining even though data often reside in relational tables. Consequently, no relational optimization has yet been possible for data mining. We exploit the transitive nature of large item sets and the so called anti-monotonicity property of support thresholds of large item sets to develop a natural *least fixpoint* operator for data mining. The operator proposed has several advantages including optimization opportunities, and traditional candidate set free large item set generation. We present an SQL3 expression for association rule mining and discuss its mapping to the least fixpoint operator developed in this paper, and thereby establish the equivalence of the top-down and bottom-up computation of large item sets in relational databases.

1 Introduction

In recent years, mining association rules has been a popular way of discovering hidden knowledge from large databases. Most efforts have focused on developing novel algorithms and data structures to aid efficient computation of such rules. While research into such procedural computation of association rules has been extensive, comparatively fewer attempts have been made to use relational machinery or SQL for *declarative* rule discovery except for a few exceptions such as [5, 9, 8, 7, 6]. Meo et al. [6] proposes an SQL like declarative query language for association rule mining. The language proposed appears to be too oriented towards transaction databases, and may not be suitable for general association rule mining. It is worth noting that association rules may be computed for virtually any type of database, transaction or not. In their extended language, they blend a rule mine operator with SQL and other additional features. The series of research reported in [9, 8, 7] mostly addressed the mining issue itself. They attempted to compute the large item sets by generating candidate sets and testing for their admissibility based on their MC model, combination, and GatherJoin operators. Essentially, these works proposed a method for implementing apriori using SQL. In our opinion by trying to faithfully copy a procedural concept into a declarative representation they retain the drawbacks and inefficiencies of apriori in the model. We are also of the opinion that the resulting SQL code is less than intuitive, unnecessarily long and complicated.

In this paper, we demonstrate that several simpler SQL3 expression exist for association rule mining that does not require candidate generation in traditional ways. We exploit several properties of transaction databases to develop a purely SQL based solution for association rule mining that uses the idea of least fix point computation. We rely on SQL3 standard as it supports complex structures including sets that we use extensively. Finally, we exploit SQL3's create view recursive construct to implement our least fixpoint operator for association rule mining.

The presentation of the paper is planned as follows. To be succinct and to avoid possible repetition, we do not actually present a discussion on association rules and the acclaimed apriori method for large item set computation. Interested readers are referred to [1, 2] for an introductory discussion as well as basic definitions pertaining to this paradigm. We begin our discussion in section 2 by presenting a discussion on declarative rule mining using SQL3 on intuitive grounds through an example. We then develop the fixpoint operator for computing large item sets in section 3.1, and present a corresponding algebraic operator ϱ in section 3.2. We present a brief discussion on an alternate method of computing large item sets in section 3.4 before we conclude in section 4.

2 Association Rule Mining Using SQL3

We begin our discussion on relational computation of association rule by presenting an example to convince the reader that such computations are actually feasible. To this direction we present a pair of SQL3 expressions for computing large item sets and association rules. First, consider a non-first normal form database, called the transaction table (or t_table{Tranid, Items}), **T** as shown in figure 1(a). Following the traditional understanding of association rule mining we expect to obtain the large item set table (l_table in figure 1(b)) and the rules table (r_table in figure 1(c)) from the source table **T** as shown once we set the support threshold at 25% (2 out of 7 transactions).

t_table

Tranid	Items
t_1	{a,b,c}
t_2	{b,c,f}
t_3	{b,f}
t_4	{a,b,c}
t_5	{b,e}
t_6	{d,f}
t_7	{d}

transaction table
(a)

l_table

Items	Support
{a}	.29
{b}	.71
{c}	.43
{d}	.29
{f}	.43
{a,b}	.29
{a,c}	.29
{b,c}	.43
{b,f}	.29
{a,b,c}	.29

large item set table
(b)

r_table

Ant	Cons	Support	Conf
{a}	{b}	0.29	1.0
{a}	{c}	0.29	1.0
{a}	{b,c}	0.29	1.0
{b}	{a}	0.29	0.40
{b}	{c}	0.43	0.60
{b}	{f}	0.29	0.40
{b}	{a,c}	0.29	0.40
{c}	{a}	0.29	0.67
{c}	{b}	0.43	1.0
{c}	{a,b}	0.29	0.67
{f}	{b}	0.29	0.67
{a,b}	{c}	0.29	1.0
{a,c}	{b}	0.29	1.0
{b,c}	{a}	0.29	0.67

association rules table
(c)

Candidate	Support
{a,b,c}	.29
{a,b,f}	$\boxed{.0}$

degree 3 candidates
(d)

Candidate	Support
{a,b}	.29
{a,c}	.29
{a,d}	$\boxed{.0}$
{a,f}	$\boxed{.0}$
{b,c}	.43
{b,d}	$\boxed{.0}$
{b,f}	.29
{c,d}	$\boxed{.0}$
{c,f}	$\boxed{.14}$
{d,f}	$\boxed{.14}$

degree 2 candidates
(e)

Candidate	Support
{a}	.29
{b}	.71
{c}	.43
{d}	.29
{e}	$\boxed{.14}$
{f}	.43

degree 1 candidates
(f)

Figure 1: (a) Transaction database **T**, (b) large item set table l_table, (c) association rules table r_table, and (d-f) the intermediate candidate sets with their supports.

To be able to develop an SQL expression that would compute the large item sets in figure 1(b), we need to examine the functional behavior of the intended expression. While there are several candidate approaches, we will discuss the one that is most familiar – apriori method, and see how we can mimic it in SQL. It will be evident that although we generate candidate sets, we do so declaratively and in a very different way. We avoid the generation of candidate sets as was discussed in [8]. The reason for avoiding the traditional way of generating the candidate sets (i.e., the ones shown in figures 1(d) through 1(f)) is that there is no simple way of generating them in a declarative fashion in the first place. The discussion and the development presented in [8] can be cited as an example.

For the sake of this discussion, we assume that several special functions are available in some implementation of SQL3 (e.g. Oracle or DB2). For example, we expect to have a function **sub** that takes three arguments, two sets of values (Items) V_1 and V_2, and a natural number k such that $|V_2| \leq k \leq |V_1|$, and returns the degree-k subsets of the set V_1 that include V_2. For any two sets S and s, s is said to be a *degree-k* subset of S if $s \in \mathcal{P}(S)$ and $|s| = k$. On the other hand, for any s such that $s \in \mathcal{P}(S)$, s is called a *distance-k* subset of S when $k = (|S| - |s|)$. The set of distance-k subsets of any set S is denoted by S^k_{sub}. Note that k ranges between 0 and $|S|$, and for any $k \geq |S|$, $S^k_{sub} = \emptyset$. Analogously, the notion of distance-k supersets can be defined. We also assume that a **sizeof** function exists that returns the cardinality of a given set. Finally, we expect to have a flatten function available in SQL3 to be able to unnest the nested columns with distinct option for the elimination of duplicates as needed. We present the following set of SQL3 expressions that compute the l_table in figure 1(b).

```
create view f_table as
   (select Items, count(Items)/m as Support
   from t_table
   group by Items);
create sequence seq increment by 1 start with 1;
create view recursive l_table as
   ((select Items, sum(Support) as Support
   from (flatten(select sub(Items, {}, 1) as Items, Support
      from f_table))
   group by Items
   having sum(Support) => δm)
   union all
   (select t.Items, sum(t.Support) as Support
   from f_table as u, (flatten distinct(select sub(f.Items, l.Items, i.Degree) as Items
      from f_table as f, l_table as l, (select Seq.Nextval as Degree
         from iteration) as i,
      where sizeof(l.Items) = i.Degree − 1 and l.Items ⊂ f.Items)) as t
   where t.Items ⊆ u.Items
   group by t.Items
   having sum(t.Support) => δm));
```

The behavior of the SQL expressions above can be explained as follows. The f_table view computes the table shown in figure 2(a). Basically, it takes the source table t_table and computes the support for each of the (distinct) item sets in it[1]. We assume that iteration{Degree} is an empty unary relation. The sequence *seq* is a counter that returns the next number in the sequence since the last call when referenced in the iteration table (in fact, from any table).

[1]Notice that $m = 7$ in the f_table view is the cardinality of t_table, and that we are using $\delta_m = .25$ as the minimum support threshold in these SQL expressions, and throughout this paper.

The l_table view has two components: the exit expression and the recursive expression. The exit expression before the union all clause creates the table shown in figure 2(b) by stripping every item set in f_table into a single item (degree-1 subsets) through the application of the **sub** function and by copying the support along. But since the **sub** function generates a set of values corresponding to each tuple, a flatten operation is required. After proper grouping and applying the selection condition (i.e., the having clause) the l_table in figure 2(c) is produced.

f_table

Items	Support
{a,b,c}	.29
{b,c,f}	.14
{b,f}	.14
{b,e}	.14
{d,f}	.14
{d}	.14

(a)

$\Longrightarrow$

Items	Support
{a}	.29
{b}	.29
{b}	.14
{b}	.14
{b}	.14
{c}	.29
{c}	.14
{d}	.14
{d}	.14
{e}	.14
{f}	.14
{f}	.14
{f}	.14

(b)

$\Longrightarrow$

l_table at step 1

Items	Support
{a}	.29
{b}	.71
{c}	.43
{d}	.29
{f}	.43

(c)

Figure 2: Computation of the exit rule.

The functionality of the recursive expression after the union all clause deserve careful explanation. The l_table generated in figure 2(c) during the exit rule computation is fed into the recursion next. The select expression takes two relations as input, the f_table, and a computed relation that is basically the set of candidate sets of cardinality k corresponding to the iteration step k. The candidate set relation is generated as follows. A "call" to the next value in the sequence *seq* is obtained through the selection of *Seq.Nextval* in the iteration table corresponding to the step k of the recursion. Notice that the sequence is initialized to 1 and hence, the first call will return a value 2 kicking off the computation of 2 item set candidates. This value is then used to select the item sets already in l_table that are potential generators of candidate sets. Notice that the condition **sizeof**$(l.Items) = i.Degree - 1$ guarantees the fact that the item sets inserted in l_table during the last iteration $(k-1)$ only will be considered. Also note that the *Degree* value is always equal to k at recursion step k. Once the candidate set generators are identified, only those tuples in f_table are selected that are supersets of l_table items. Then the candidates are generated through the application of **sub** function as before using l_table and f_table items, and the iteration step k. But this time flatten is applied with distinct option to eliminate all duplicates. That means we only generate all distinct candidates without their support (unlike the first step). This is shown in figure 3(b). Once the candidates are known, the outer select expression computes the candidate new entries for l_table shown in figure 3(d) with their supports. The set of tuples in 3(d) meeting the minimum threshold are then added to the recursive table l_table as shown in figure 3(c), and the computation goes on until a fixpoint is reached.

A careful examination will show that the way the candidate sets are generated are quite different from traditional approaches. For example, a smart traditional item set generator only looks at the large item sets generated at step k to generate candidate sets for step $k+1$. From figure 1(e), it is clear that item set {b,c,f} cannot be a candidate as {c,f} is not a large item set (support of all non-large item sets are marked with a box in figure 1), even though {b,f} is a large item set. This determination requires a test and a cross examination to see if any other subset (say, {c,f}) of the candidate (say, {b,c,f}) is not a large item set when the generation of a candidate set ({b,c,f}) is being considered based on any large item set ({b,f}). Note that we can easily get confused and consider {b,c,f} to be a candidate because {b,f} and {c} are both large.

Although the end results in both the cases, apriori and our method, are identical, notice that l_table from step 1

Items	Support
{a}	.29
{b}	.71
{c}	.43
{d}	.29
{f}	.43

(a)

⇒

Items
{a,b}
{a,c}
{b,c}
{b,e}
{b,f}
{c,f}
{d,f}

(b)

⇒

Items	Support
{a,b}	.29
{a,c}	.29
{b,c}	.29
{b,c}	.14
{b,e}	.14
{b,f}	.14
{b,f}	.14
{c,f}	.14
{d,f}	.14

(c)

⇒

new l_table at step 2

Items	Support
{a}	.29
{b}	.71
{c}	.43
{d}	.29
{f}	.43
{a,b}	.29
{a,c}	.29
{b,c}	.43
{b,f}	.29

(d)

l_table from step 2

Items	Support
{a}	.29
{b}	.71
{c}	.43
{d}	.29
{f}	.43
{a,b}	.29
{a,c}	.29
{b,c}	.43
{b,f}	.29

(e)

⇒

Items
{a,b,c}
{b,c,f}

(f)

⇒

Items	Support
{a,b,c}	.29
{b,c,f}	.14

(g)

⇒

new l_table at step 3

Items	Support
{a}	.29
{b}	.71
{c}	.43
{d}	.29
{f}	.43
{a,b}	.29
{a,c}	.29
{b,c}	.43
{b,f}	.29
{a,b,c}	.29

(h)

Figure 3: Computation inside the recursion.

we did not generate the candidate set {a,b,f} shown in figure 1(d) while we have generated {b,c,f} shown in figure 3(f). This is because we did not cross check just to avoid excessive cost of computing. But if we did, we could have avoided the generation of useless candidate sets such as {b,c,f}. This is possible because we look ahead and generate the candidates from both the database entries and the large item sets generated at each recursive step.

Once the large item sets are at hand, rule generation becomes a relatively simple task for any given confidence threshold η_m. We present the following SQL3 view that computes all the association rules shown in figure 1(c) that are entailed by **T**.

```
create view r_table as
  (select a.Items, c.Items\a.Items, c.Support,
     c.Support/a.Support
  from l_table as a, l_table as c
  where a.Items ⊂ c.Items and c.Support/a.Support ≥ η_m)
```

3 Relational Operators for Data Mining

For the sake of this discussion, and without loss of any generality, let us consider two relation schemes $r\{$*Items, Support*$\}$ and $s\{$*Items, Support*$\}$ such that *Items* represents item sets and *Support* represents the percentage repetition frequency of the item set patterns in some transaction database[2].

We begin our discussion by introducing a few necessary definitions. First, let *k-sub* be a function that for any given item set s and k such that $1 \leq k \leq |s|$, returns all distance-k subsets of s, i.e., *k-sub*$= S^k_{sub}$. Using this function we define the distance-k subsets of an item set relation as follows.

Definition 3.1 Let r be a relation such that its scheme includes a set valued attribute *Items*. Then for any $k \geq 0$, the distance-k subset of r is defined as

$$\xi^k_{Items}(r) = \{t \mid \exists u, v(u \in r \wedge v \in k\text{-}sub(u[Items]) \wedge t = \langle v, u[R \setminus Items]\rangle)\}$$

[2]It is worth mentioning here that this restriction is artificial, and is not necessary in practice. In reality, we could allow any SQL3 compliant table as we have already demonstrated in section 2. We consider such a scheme here for simplicity only.

Finally, we define an *item set join*, $\bowtie^i$, operator as follows. This operator allows the selection of next level candidate frequent large item sets in a chain of item sets.

Definition 3.2 Let r and s be two relations on scheme {Items, Support}. Also let dis^1 be a binary Boolean function that returns true if for any two item sets s' and s'', s'' is a distance-1 subset of s', false otherwise, where s' is the first argument. Then the distance-k item set join of r and s is defined as

$$r \bowtie^i s = \{t \mid t \in r \wedge \exists u(u \in s \wedge sub^1(t[Items], u[Items]) = true)\}$$

Notice that $r \bowtie^i s$ returns all tuples in r such that there is a tuple in s that has an item set for which the item set in tuple of r is a distance-1 superset. Other tuples are not selected.

3.1 Least Fixpoint Operator

We are now ready to present our least fixpoint operator $\mathbf{T}_{\mathcal{D}}$ for computing the frequent large item sets from a database $\mathcal{D}$. In this paper, we define the operator in a very informal way without strict accuracy just to get our ideas across. A more precise development may be found in an extended paper. But, before we present the fixpoint operator, we need to present the idea of the consequence of a transaction database, and its interpretation.

Definition 3.3 (Interpretations of $\mathcal{D}$) Let $\mathcal{D}\{Items, Support\}$ be a transaction database. Let $con(\mathcal{D}) = \bigcup_{k=1}^{n} \xi_{Items}^{k}(\mathcal{D})$, where n is the maximum cardinality of the item set patterns in $\mathcal{D}$. Then, I, called an interpretation of $\mathcal{D}$, is any set $I \subseteq_{Items} \mathcal{G}_{Support=sum(Support)}(con(\mathcal{D}))$. For any I, ${}_{Items}\mathcal{G}_{Support=sum(Support)}(con(\mathcal{D}))$ is called a consequence of I[3]. Let $\mathcal{I}$ be the set of all possible interpretations of $\mathcal{D}$. □

Definition 3.4 (Model of $\mathcal{D}$) Let I be an interpretation of a database $\mathcal{D}$. Then, I is a model of $\mathcal{D}$, denoted $M_{\mathcal{D}}$, iff the consequence of I is I itself, i.e., $con(I) = I$. □

Definition 3.5 (Immediate Consequence Operator) Let $\mathcal{D}$ be a transaction database, I be any interpretation, $\mathcal{N}$ be a set of natural numbers, and δ_m be any support threshold. We define $\mathbf{T}_{\mathcal{D}}$ to be the immediate consequence operator with respect to a natural number k such that $\mathbf{T}_{\mathcal{D}} : \mathcal{P}(\mathcal{I}) \times \mathcal{N} \mapsto \mathcal{P}(\mathcal{I})$, and that

$$\begin{aligned}
\mathbf{T}_{\mathcal{D}}(I,k) &= \sigma_{Support \geq \delta_m}({}_{Items}\mathcal{G}_{Support=sum(Support)}(\xi_{Items}^{k}(\mathcal{D}))), \text{ if } k=1,\\
\mathbf{T}_{\mathcal{D}}(I,k) &= \sigma_{Support \geq \delta_m}({}_{Items}\mathcal{G}_{Support=sum(Support)}((\xi_{Items}^{k}(\mathcal{D})) \bowtie^i I)), \text{ otherwise.}
\end{aligned}$$

The above definition suggests that the operator $\mathbf{T}_{\mathcal{D}}$ is a monotone as it produces a set of tuples which consist of distance-k supersets of item sets in I along with their total supports, such that the sets produced by $\mathbf{T}_{\mathcal{D}}$ are monotonically increasing sequences. That is, if we apply $\mathbf{T}_{\mathcal{D}}$ to $\mathbf{T}_{\mathcal{D}} \uparrow^k$, we hope to obtain a larger set, and thus we can expect to reach a fixpoint. In other words, successive applications of $\mathbf{T}_{\mathcal{D}}$ on an interpretation is an increasing sequence of interpretations. Notice that this sequence is bounded by the size of the item sets, as that is the maximum cardinality of any item set that can occur. Consequently, the computational characteristics of a model for $\mathcal{D}$ can be captured as follows.

Theorem 3.1 (Model of $\mathcal{D}$ as Fixpoint of $\mathbf{T}_{\mathcal{D}}$) Let $\mathcal{D}$ be a database and I be any interpretation. Then, I is a model of $\mathcal{D}$, denoted $M_{\mathcal{D}}$, iff $\mathbf{T}_{\mathcal{D}}(I,k) \subseteq I$, for some non-zero k such that for all $j > k$, $\mathbf{T}_{\mathcal{D}}(I,j) = \mathbf{T}_{\mathcal{D}}(I,k)$. □

[3]Note that $\mathcal{D}$ itself is a consequence of $\mathcal{D}$.

The condition $\mathbf{T}_{\mathcal{D}}(I,j) = \mathbf{T}_{\mathcal{D}}(I,k)$ in the definition above is important because an interpretation I may satisfy the condition $\mathbf{T}_{\mathcal{D}}(I,k) \subseteq I$ yet not be a model. For example, if we consider a set of tuples that are derived from only distance-j subsets of an item set such that $j > k$. In that case $\mathbf{T}_{\mathcal{D}}(I,k)$ will produce nothing, and hence I will be accepted as a model, while for some $m > j > k$, $I \subseteq \mathbf{T}_{\mathcal{D}}(I,m)$. Hence, we need to find the minimum k for which $\mathbf{T}_{\mathcal{D}}(I,k)$ does not produce anything new, and similarly for any $j > k$.

The bottom-up fixpoint computation can now be defined as follows:

$$\begin{aligned}
\mathbf{T}_{\mathcal{D}}\uparrow^{0} &= \emptyset \\
\mathbf{T}_{\mathcal{D}}\uparrow^{k+1} &= \mathbf{T}_{\mathcal{D}}(\mathbf{T}_{\mathcal{D}}\uparrow^{k}) \\
\mathbf{T}_{\mathcal{D}}\uparrow^{\omega} &= \bigcup_{m\leq\omega\leq n} \mathbf{T}_{\mathcal{D}}\uparrow^{m}
\end{aligned}$$

where ω is a limit ordinal. Fortunately ω is no greater than the maximum possible cardinality of the item sets in $\mathcal{D}$, i.e., no more than $n = |\mathcal{I}|$, where $\mathcal{I}$ is the set of all possible items in the transaction database $\mathcal{D}$.

There is a serious subtlety we must point out here. For the computation to go ahead correctly, and the model computed precisely, it is important to compute $\mathbf{T}_{\mathcal{D}}$ in increasing sequence without any break in the sequence or repeating a step, because of the way $\mathbf{T}_{\mathcal{D}}$ and its associated machinery is defined. Suppose, if we repeat a step, $j < k$ once $\mathbf{T}_{\mathcal{D}}\uparrow^{k}$ is computed, then unwanted distance-j subsets of certain item sets could be introduced in I wrong support computation would result. This is because the support counts are derived from existing tuples and added to find the total count. The functions do not memorize the components from which the total is derived, and hence any new introduction will be treated as a fresh and valid entry and will be added to the total.

The fixpoint $\mathbf{T}_{\mathcal{D}}$ can also be computed using the following algorithm.

```
algorithm: fixpoint;
input: A transaction table $\mathcal{D}$, support threshold $\delta_m$.
output: Model $M_{\mathcal{D}}$.
begin
   initialize $C$ as $C \leftarrow \sigma_{Support\geq\delta_m}({}_{Items}\mathcal{G}_{Support=sum(Support)}(\xi^{1}_{Items}(\mathcal{D})))$.
   initialize $P$ as empty.
   while $(C \setminus P \neq \emptyset)$ do
      compute $P \leftarrow C$.
      assign $k = k + 1$.
      compute $C \leftarrow C \cup (\sigma_{Support\geq\delta_m}({}_{Items}\mathcal{G}_{Support=sum(Support)}((\xi^{k}_{Items}(\mathcal{D})) \bowtie^{i} C)))$.
   compute $M_{\mathcal{D}} \leftarrow C$.
end.
```

It is easy to show that the algorithm above computes the fixpoint defined in terms of $\mathbf{T}_{\mathcal{D}}$, and hence the result below follows.

Theorem 3.2 For any transaction database $\mathcal{D}$, the models computed by the algorithm **fixpoint** and the fixpoint of $\mathbf{T}_{\mathcal{D}}$ are identical. □

It is also possible to show that, for a given database and a support threshold, every large item set computed by apriori like algorithms is a member of the model $M_{\mathcal{D}}$ computed by algorithm **fixpoint**, and vice versa. Consequently, it is also easy to establish the equivalence of such top-down computation (apriori) with the bottom-up computation ($\mathbf{T}_{\mathcal{D}}$) through model construction presented in this paper.

3.2 A Relational Fixpoint Operator for Frequent Item Sets

We abstract the functionalities of $\mathbf{T}_{\mathcal{D}}$, or equivalently the procedure **fixpoint** in a simple yet intuitive operator as follows.

Definition 3.6 (Fixpoint Operator) Let It and Su be two column names in a transaction database $\mathcal{D}$ with corresponding item set and support type domains, δ_m be the minimum support threshold, and n be the maximum cardinality of the item sets in It of $\mathcal{D}$. Then the large item sets operator ϱ is defined as follows:

$$\begin{aligned}\varrho^1_{It,Su}(\mathcal{D}) &= \sigma_{Su\geq\delta_m}({}_{It}\mathcal{G}_{Su=sum(Su)}(\xi^1_{It}(\mathcal{D})))\\ \varrho^n_{It,Su}(\mathcal{D}) &= \varrho^{n-1}_{It,Su}(\mathcal{D}) \cup (\sigma_{Su\geq\delta_m}({}_{It}\mathcal{G}_{Su=sum(Su)}((\xi^n_{It}(\mathcal{D})) \bowtie^i \varrho^{n-1}_{It,Su}(\mathcal{D}))))\end{aligned}$$

Notice that this recursive version of the fixpoint operator is equivalent to the iterative version presented in the algorithm **fixpoint**. The correctness of the fixpoint operator is captured through the following theorem.

Theorem 3.3 ϱ and $\mathbf{T}_{\mathcal{D}}$ compute identical models. □

3.3 Optimization Opportunities

We would just like to point out here that there exists many opportunities for optimization in all the expressions and algorithms just presented to speed up the query processing. It is also possible to further speed up the query processing by developing new indexing techniques for set valued attributes such as item sets, to be able to find tuples that contain a subset or superset of a given set, and then use this index to implement $\bowtie^i$ operator as discussed earlier. Recall that $\bowtie^i$ require such a join based on superset relationships.

3.4 An Alternative SQL3 Approach

While we have presented an apriori like approach in this paper to develop the relational fixpoint operator for data mining, it is possible to mine any nested relational database without any need for extended operators. Since the scope of this paper prohibits a detailed discussion on this topic, we will briefly outline a series of SQL3 expressions that can be used to compute the large item sets in a more declarative way that does not require such specialized operators. However, this declarative method does not utilize the anti-monotonicity property and hence, if implemented as is, may have some performance related drawbacks compared to other known and efficient methods. But the purpose of this discussion is to highlight the fact that, several SQL3 expressions actually exist for the computation of association rules, and it may become possible for a query optimizer to select the best possible strategy for the computation of these expressions depending on the database states.

We present below three view definitions in SQL3 that can be used to compute the large item sets for the database **T** introduced in section 2. The corresponding views in these expressions are shown in the figure 4(a) through 4(d). Readers may verify that the table produced by the view l_table in figure 4(d) is "equivalent" to the table l_table shown in figure 1(b). In this method, first we use the f_table introduced earlier in section 2 to compute the so called *intersection table* defined below. The computation of the intersection table is somewhat reminiscent of the closed set [10] and equivalence class [3] computation, yet it is quite different. The main distinction is that this approach allows us to develop the declarative expression we present below that was not quite possible until now without this observation.

```
create view recursive int_table as
  ((select distinct intersect(t.Items, p.Items), .0
  from f_table as t, f_table as p
  where $t.Items \not\subseteq p.Items$ and $p.Items \not\subseteq t.Items$
    and not exists (select *
      from f_table as f
      where f.Items = intersect(t.Items, p.Items)))
  union
  (select distinct intersect(t.Items, p.Items), .0
  from int_table as t, int_table as p
  where $t.Items \not\subseteq p.Items$ and $p.Items \not\subseteq t.Items$
    and not exists (select *
      from f_table as f
      where f.Items = intersect(t.Items, p.Items))))
```

Once the intersection table is at hand, the support count and large item set computation is fairly straight forward. But notice that the l_table thus computed is not identical to the one computed in figure 1(b). We claim that they are equivalent because the table 1(b) can be inferred from table 4(d). In other words, table 1(b) contains redundant tuples that are implied by table 4(d). Consequently, the rules generated from table 4(d) are also non-redundant. Interested readers are referred to [4] for an in depth analysis of this approach that will appear elsewhere.

```
create view i_table as
  (select t.Items, p.Support
  from f_table as p,
      ((select *
      from f_table)
      union
      (select *
      from int_table)) as t,
  where $t.Items \subset p.Items$)

create view c_table as
  (select t.Items, sum(t.Support) as Support
  from ((select *
      from f_table)
      union
      (select *
      from i_table)) as t
  group by t.Items)

create view l_table as
  (select Items, Support
  from c_table
  where $Support \geq \delta_m$)
```

int_table

Items	Support
{b}	.0
{f}	.0
{b,c}	.0

(a)

i_table

Items	Support
{b}	.29
{d}	.71
{f}	.38
{b,c}	.29
{b,f}	.38

(b)

c_table

Items	Support
{b}	.29
{d}	.71
{f}	.38
{b,c}	.29
{b,e}	.38
{b,f}	.29
{d,f}	.29
{a,b,c}	.38
{b,c,f}	.29

(c)

l_table

Items	Support
{b}	.71
{d}	.29
{f}	.38
{b,c}	.38
{b,f}	.29
{a,b,c}	.29

(d)

Figure 4: Set of computed views from the database **T**: an alternative approach.

4 Conclusions

The goal of this paper was two fold. First, to demonstrate that SQL3 can be used to mine relational databases in order to discover association rules. Second, to introduce a relational fixpoint operator for large item set computation. With the help of the machineries developed in this paper, we have been able to show that a bottom-up method for the computation of association rules is possible that is equivalent to its top-down counterpart. The novelty of the work reported in this paper lies in its simplicity and formal foundation of the approach. The alternative method proposed for the large item set relation computation acts as the key for the development of yet another simple top-down procedure for rule induction much like a theorem prover in logic programming paradigm.

Our current and future research in this area focuses on the development of query optimization strategies and an efficient implementation of the proposed operators. We are working on new indexing techniques for set valued attributes for non-first normal form databases so that efficient join algorithms can be developed for data mining applications. Developing optimization techniques for aggregate queries is also another research issue we plan to address in our future research.

References

[1] Rakesh Agrawal, Tomasz Imielinski, and Arun N. Swami. Mining association rules between sets of items in large databases. In Peter Buneman and Sushil Jajodia, editors, *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 207–216, Washington, D.C., May 26-28 1993.

[2] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo, editors, *Proceedings of 20th International Conference on Very Large Data Bases*, pages 487–499, Santiago, Chile, September 12-15 1994.

[3] Yves Bastide, Rafik Taouil, Nicolas Pasquier, Gerd Stumme, and Lotfi Lakhal. Mining frequent patterns with counting inference. *SIGKDD Explorations*, 2(2):66–75, 2000.

[4] Hasan M. Jamil. Surprise! SQL is enough for data mining. Technical report, November 2000.

[5] Rosa Meo, Giuseppe Psaila, and Stefano Ceri. A new SQL-like operator for mining association rules. In T. M. Vijayaraman, Alejandro P. Buchmann, C. Mohan, and Nandlal L. Sarda, editors, *Proceedings of 22nd International Conference on Very Large Data Bases*, pages 122–133, Mumbai, India, September 3-6 1996.

[6] Rosa Meo, Giuseppe Psaila, and Stefano Ceri. An extension to SQL for mining association rules. *Data Mining and Knowledge Discovery*, 2(2):195–224, 1998.

[7] Karthick Rajamani, Alan Cox, Bala Iyer, and Atul Chadha. Efficient mining for association rules with relational database systems. In *Proceedings of the International Database Engineering and Applications Symposium*, pages 148–155, 1999.

[8] Sunita Sarawagi, Shiby Thomas, and Rakesh Agrawal. Integrating mining with relational database systems: Alternatives and implications. In Laura M. Haas and Ashutosh Tiwary, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 343–354, Seattle, Washington, June 2-4 1998.

[9] Shiby Thomas and Sunita Sarawagi. Mining generalized association rules and sequential patterns using SQL queries. In *Proceedings of the sixth ACM SIGKDD international conference on knowledge discovery and data mining*, pages 344–348, New York, NY, August 1998.

[10] Mohammed Javeed Zaki. Generating non-redundant association rules. In *Proceedings of the sixth ACM SIGKDD international conference on knowledge discovery and data mining*, pages 34–43, Boston, MA USA, August 2000. ACM Press.

Monitoring Change in Mining Results

Steffan Baron[1] and Myra Spiliopoulou[2]

[1] Institut für Wirtschaftsinformatik, Humboldt-Universität zu Berlin
sbaron@wiwi.hu-berlin.de
[2] Institute of Business and Technical Information Systems,
Otto-von-Guericke Universität Magdeburg
myra@iti.cs.uni-magdeburg.de

Abstract. In the last years the datasets available have grown tremendously, and the development of efficient and scalable data mining algorithms has become a major research challenge. However, since the data is more dynamic than static there is also a strong need to update previously discovered rules and patterns. Recently, a couple of studies have emerged dealing with the topic of incremental update of discovered knowledge. These studies mostly concentrate on the question whether new rules emerge or old ones become extinct.
We present a framework that enables the analyst to monitor the changes a rule may undergo when the dataset the rules were discovered from is updated, and to observe emerging trends as data change. We propose a generic rule model that distinguishes between different types of pattern changes, and provide formal definitions for these. We present our approach in a case study on the evolution of web usage patterns. These patterns have been stored in a database and are used to observe the mining sessions as snapshots across the time series of a patterns lifetime.

1 Introduction

While there are many sophisticated techniques for the discovery of knowledge in a large range of domains, methods for knowledge maintenance and updating are emerging at a slower pace. In recent years a considerable number of studies aiming on the update of previously discovered knowledge have emerged. Most of those works focus on the refreshment of the mining results when the underlying dataset is updated. They investigate methods that actualize the mining results on the basis of the data update only, i.e., without scanning the whole dataset again. More recent works also consider the *lifetime* of a rule, i.e., the time interval in which it is sufficiently supported by the data.

In this study, we propose a framework for updating mining results *and* observing their evolution, thus effectively combining both research trends in a general framework. Our work is motivated by the need for sustainable knowledge: beyond updating knowledge, we need a mechanism monitoring its evolution. For example, consider an association rule "Product A $\Rightarrow$ Product B" appearing in multiple consecutive mining sessions. A decrease in the confidence of this rule would imply that the two products are less strongly correlated than before. To observe this decrease, we propose the modeling of rules as temporal objects and of their statistics as time series of snapshots corresponding to the mining sessions. Our work differs from other approaches in this domain in a number of aspects. First, rather than only investigating techniques to discover rules that have newly emerged or become extinct after an update, we focus on the changes a specific rule may undergo through several updates to the dataset. We model rules as temporal objects, on which techniques for time series analysis can be applied to discover trends and trend similarities. Second, we propose a generic rule model appropriate for more than one rule type. Third, we consider two aspects of rule evolution, namely changes in the statistics of a rule, as in the above example, and changes in the content of a rule. Based on these types of change, we give formal definitions of possible types of pattern evolution.

The article is organized as follows. In the next section, we give an overview of related work. In Sec. 3, we present our framework, in which rules are modeled as temporal objects. We then distinguish between different types of pattern evolution. In Sec. 4, we demonstrate the advantages of our framework in a case study on monitoring user navigation patterns. Sec. 5 summarizes the study and gives a brief outlook on future work.

2 Related Work

The earliest incremental mining algorithms have been designed for the refreshment of association rules. This mining paradigm is still the focus of the majority of contributions in this domain. The aim is to avoid a complete re-run of the mining algorithm on the whole dataset when new data is added. The most notable works in this area are by Cheung et al. [6, 5]. They deal with the problem of maintaining discovered rules after updates on the original database. An algorithm FUP_2 is proposed which uses information from previous mining sessions to maintain rules after *insertions* and *deletions* in the original dataset. The basic idea is that an itemset X may only remain large after an update to the original database, if it is large in both the original database and in the update. Ayan et al. propose an algorithm *UWEP* which also deals with the incremental update of association rules [1]. Similar to FUP_2 algorithm the aim is the efficiency in performing the update. The main performance advantage of *UWEP* is achieved by pruning away the supersets of a large itemset in DB as soon as they are known to be small in the updated database $DB + \delta$. This methodology yields a much smaller candidate set especially when the set of new transactions does not contain some of the old large itemsets. In [11], an incremental version of the well known *Partition* algorithm for association rules discovery is proposed. Basically, DB is used as one partition, and δ is used as another partition. Then, a slight variation of the original algorithm is used to update the patterns. In this study, efficiency is achieved by minimizing the number of scans over DB and δ. Thomas et al. propose an algorithm which reuses not only the large itemsets from previous mining sessions but also their *negative borders*, the candidate itemsets which did not have enough support to become large [14]. A similar approach was developed independently by Feldman et al. at about the same time [8].

There have also been studies based on other mining paradigms. Comparable work which emphasizes on dynamic sequential data has been done by Wang et al. [15]. They present a time independent algorithm for the incremental discovery of patterns in large and dynamic sequential data which takes advantage of already discovered patterns and computes the change by accessing only the affected part of the dataset. Ester et al. propose a similar approach for clustering in a data warehouse environment [7]. They present an incremental clustering algorithm, based on DBSCAN, that examines which part of an existing clustering is affected by an update of the database and adjusts the clusters accordingly. For a detailed description of these studies see the general overview of pattern evolution in [2].

Another part of related work focuses more on the similarity between rules and on the temporal aspects of rules. Ganti et al. compute a deviation measure which makes it possible to quantify the difference between two datasets in terms of the model they induce, called `FOCUS` [9]. A new notion of surprising temporal patterns and algorithms to compute these is proposed by Chakrabarti et al. [3]. They argue that once the analyst is already familiar with prevalent patterns in the data, the greatest incremental benefit is likely to be from changes in the relationship between item frequencies over time. [4] concentrate on the identification of interesting temporal features like valid period or periodicity of association rules. The first two approaches take similarity of rules and changes a rule may undergo into account, the latter approach temporal properties of a rule. But these two aspects of a rule belong together, and should not be considered separately. Therefore, we model both aspects as a time series of changes a rule is subjected to in time. In a recent work Ganti et al. introduce a new dimension which takes the temporal aspect of the collected data into account, called *data span dimension* [10]. Here, the analyst can specify an arbitrary time window from which the data is taken for the analysis. Also they describe new model maintenance algorithms with respect to the selection constraints on the data span dimension for frequent itemsets and clustering.

The main difference to our contribution is that they consider a certain time span, but only at a fixed point in time. The analyst can choose to mine all the data available so far, or a block of data that is more suited for the purpose of the analysis. But the analysis itself is limited to that fixed time point treating the analyzed data as a whole, possibly overseeing interesting changes of a rule at the end of the specified time window, for example.

3 A Framework for Monitoring Pattern Evolution

As mentioned above there are two different types of rule change. First, the statistics of a rule may change. For example, an association rule $X \Rightarrow Y$ may hold with less confidence after adding the increment database δ because a smaller percentage of transactions in the updated database $DB + \delta$ that contain X also contain Y. Second, the content of a rule may change. For example, after updating the database an association rule may become less restrictive by having more items in the consequent than before, or a user navigation pattern may follow a different path, though leading from the same start page to the same end page.

3.1 A Generic Rule Model

We model in a rule both its content and its statistics and, along with the timestamp of the mining session which has produced it and a key which uniquely identifies the rule across consecutive mining sessions. Then, a rule is a temporal object with the following signature:

$$R = (ID, query, timestamp, stats, body, head)$$

Here, ID is the unique identifier of rule R which is built from *body* and *head* of that rule, *query* is the mining query which has produced it at the given *timestamp*, and *body* and *head* describe the antecedent and the consequent of R, respectively. For the statistics of a rule, denoted by *stats*, we need a special treatment because some values refer to the entire rule, whereas other values only refer to a specific item. For example, in a rule $X \Rightarrow Y$ the support of the single item X or Y may also be of interest. In order to overcome this problem, we model the statistics concerning a single item with the item as part of the body or head of a rule directly in the rules content, and denote only statistics which refer to the rule as a whole by *stats*.

Example. In the simplest case of association rule discovery, a query Q specifies a lower boundary on the permitted support and an upper boundary on the rule length. We observe a mining session as the run of a mining query over the dataset, producing a quite large set of rules in the general case. Using the above notation, an association rule $A \Rightarrow B$ with support $s = 10\%$ and confidence $c = 60\%$ produced by query Q at time stamp t_0 would be modeled as $R_0 = (ID_{AB}, Q, t_0, [s = 10\%, c = 60\%], A, B)$ where ID_{AB} denotes an identifier that accompanies the rule across all mining sessions and makes it uniquely identifiable, as long as body and head of the rule does not change. □

3.2 Different Types of Patterns

A rule base does not consist solely of association rules. A sequence miner outputs frequent sequences, while a clustering algorithm computes clusters of similar items. When modeling a rule base, these types of rules should also be taken into account.

Without loss of generality, for frequent sequences we define the whole sequence but the last item to be the body of the rule, and the last item to be the head of the rule. Then, a frequent sequence $ABCDE$ is modeled as $ABCD \Rightarrow E$. The statistics for each item are directly stored in the item, i.e., in the body or head of the rule. The same yields for more complex sequences like web usage patterns. As a first step, the start page and the end page of a navigation can be modeled as an association rule of the form *startpage* $\Rightarrow$ *endpage*. However, since this leads to an information loss, the actual navigation pattern is decomposed into paths which can be treated just as sequences. For example, consider a frequent navigation pattern starting in page `S.html` and ending in `E.html` which consists of two paths. On the first path, users accessed two pages `T1.html` and `T2.html` on their way from `S.html` to `E.html`, on the second path, they visited one and the same page `T3.html`. This navigation pattern is modeled as two sequential patterns `S`, `T1`, `T2` $\Rightarrow$ `E` and `S`, `T3` $\Rightarrow$ `E`.

For clusters we use a similar approach: each cluster is identified by its centroid and is decomposed into its constituent items by giving the value of each item, along with the centroid of the cluster it belongs to and the distance to the centroid. The advantage of this approach is that it also works for a density based clustering. Then, a cluster is a set of association rules of the form $item, centroid, distance \Rightarrow cluster$. However, as with complex navigation patterns, rather than a single item the body and head of a rule would be a set of items. Therefore, we extend the generic rule model to subsume also more complex types of patterns:

$$R = (ID, query, timestamp, stats[\,], body[\,], head[\,])$$

Then, a simple association rule with a single item in the antecedent and a single item in the consequent is just modeled as $body[0] \Rightarrow head[0]$.

3.3 Monitoring Pattern Change

As described in Sec. 1, we consider two types of changes that may affect a pattern: changes to its statistics and changes to its content. Regarding changes to the statistics of a rule, we give the following definition for the *evolution* of a rule:

Definition 1 *As* evolution *of a rule we denote changes to its statistical measurements in time.*

When monitoring a rules statistics we need to identify the rule non-ambiguously through several mining sessions. For this purpose, a unique key is derived from body and head of the rule. Hence, if the content of the rule changes from one mining session to another, the identifier also changes, and we say the rule has died:

Definition 2 *Changes to the contents of a rule and changes to the statistics of a rule that violate given thresholds lead to its* death.

However, this approach may be considered too restrictive since it circumvents monitoring of changes to the contents of a rule. When monitoring content changes the above definitions are not sufficient:

Definition 3 *As* mutation *of a rule we denote changes to the contents of a rule provided that given thresholds are not violated.*

However, a global identifier is still needed. For example, consider an association rule $AB \Rightarrow CD$ which is generated from the initial dataset. After an update we apply some incremental mining algorithm, and obtain the rule $AB \Rightarrow CE$. Here, the question arises whether there has emerged a new rule or if the old rule has mutated. Only if each rule is uniquely identifiable, we can answer this question. To derive the identifier, the analyst could decide to use only the rules body, with the effect that one identifier may represent more than one rule, i.e., all rules with the same body. The question of how such an identifier can be produced is the subject of future work.

Now we consider the case of a rules death. When it dies, it is still important to know why it disappeared. In general, the analyst specifies minimum thresholds for support and confidence to observe patterns from data. Then, all rules that satisfy the given thresholds are found. After an update, it is possible for a rule to vanish if it does not fit the requirements defined by the analyst. For monitoring these rules three different approaches are reasonable. First, exactly those threshold values given by the analyst are used to discover rules. This would imply that the analyst needs to adjust the threshold values and re-run the mining algorithm to locate those rules that have vanished. Second, internally lower values for support and confidence as those given by the analyst are used to discover additional rules. These rules are then used to locate rules that disappeared through an incremental mining run, as opposed to external rules that are presented to the analyst. Finally, the extreme is not to restrict support and confidence, and produce all possible rules from the dataset. However, again only those rules are presented to the analyst that fulfill the given requirements. On the other hand, it is also valuable to monitor emerging rules. Then, a complete time series for the entire lifespan of a pattern can be derived.

4 Case Study

In order to prove our concepts we investigated the evolution of frequent sequences as produced by the Web Usage Miner (WUM), a tool for analyzing the behavior of a web sites visitors developed at Humboldt-University Berlin [12]. Using WUM, the analyst can specify a template expressed in the MINT query language which describes the navigation patterns he is interested in [13]. Moreover, the number of discovered patterns can be limited by giving constraints on support and confidence for the pages visitors accessed throughout their navigation.[1] The results WUM produces are generalized sequences (*g-sequences*) which are instances of the template given by the analyst, where each g-sequence consists of a non-empty set of navigation patterns [12].

The dataset used for the case study is a common access log file as produced by a web server spanning seven months. From the data three equally spaced samples were drawn, each representing a single month. All page names in the log file were mapped to a concept hierarchy, where each concept represents pages

[1] For a complete example see `http://wum.wiwi.hu-berlin.de`.

with a particular subject on the web server. Then, rather than navigation patterns between single pages navigation patterns within these concepts have been analyzed. For the analysis we used the first and the last approach as described in Sec. 3.3.

4.1 The Constraint-Based Approach

In the first mining run the following MINT query was issued which generated all navigation patterns up to length 5 where the start page has an absolute support of more than 100 visitors and the end page a minimum support of more than 50 visitors within the same navigation pattern, and more than 20% of the users started in page `x` should also visit page `y`.

```
SELECT t
 FROM NODE AS x y,
 TEMPLATE x [0;5] y AS t
 WHERE x.support > 100
 AND y.support > 50
 AND ( y.support / x.support ) > 0.2
```

The results comprise a number of g-sequences that match the given template. A sample of results is shown in Fig. 1. Each g-sequence consists of a pattern identifier which accompanies the pattern uniquely within the mining session, an antecedent (the start page of a navigation) and a consequent (the end page of a navigation). Antecedent and consequent consists of the page identifier, the page name (the concept name in our example), the occurrence (how often this page was accessed), and the absolute number of visitors accessed this page within their navigation. A g-sequence is uniquely identified across all mining sessions by its antecedent and consequent, i.e., the concept name along with its occurrence.

The results of the mining run were imported into a relational DBMS according to our generic rule model. Fig. 2 shows the sample of g-sequences depicted in Fig. 1 as stored in the database. For simplicity, the results were not normalized before importing them in the DBMS. Instead, all attributes of a pattern were stored in one single relation, i.e., in one separate relation for the results from each month, where the key of the relation consists of four attributes, `ant_page`, `ant_occ`, `con_page` and `con_occ`. The values for the confidence attribute were computed during the import.

```
PatternID=1: [3000016;I143;2;132], [3000016;I143;3;93]
PatternID=2: [3000006;I111;1;738], [3000013;I114;1;153]
PatternID=3: [3000002;I13;1;274], [3000004;I135;1;69]
PatternID=4: [3000008;I11;1;320], [3000006;I111;1;143]
PatternID=5: [3000013;I114;1;407], [3000013;I114;2;172]
```

Fig. 1. Sample of g-sequences as produced by WUM.

ant_page	ant_occ	con_page	con_occ	ant_supp	con_supp	conf
I143	2	I143	3	132	93	0.7045
I111	1	I114	1	738	153	0.2073
I13	1	I135	1	274	69	0.2518
I11	1	I111	1	320	143	0.4469
I114	1	I114	2	407	172	0.4226

Fig. 2. Sample of g-sequences as stored in the DBMS.

In order to identify patterns that have emerged or disappeared through the different mining sessions, several SQL queries were issued. We found 8 patterns that disappeared and 3 patterns that emerged of a total of 27 patterns during the second mining session. Through mining the third dataset 5 patterns disappeared and 3 patterns emerged. After that, we investigated the statistics of those patterns. Fig. 3 is a time series of the statistics of a sample pattern which was drawn over the different mining sessions. The first diagram shows a rising support of the antecedent of the pattern. However, since the support of the consequent does not rise equally we encounter a loss of confidence for that pattern.

Summarizing our results, it turned out that the differences which led to changes in the result sets were very small. Therefore, we continued to analyze the changes that might be interesting using the third approach mentioned in Sec. 3.3.

4.2 The Unlimited Approach

This approach is mainly based on the idea by Chakrabarti et al. mentioned in Sec. 2 [3]. We extend this idea by introducing a *difference* parameter for each statistical measurement which can be used to identify interesting changes. However, since also strong changes of patterns that do not have minimum support and confidence might be interesting, we applied this parameter on each pattern discovered from the dataset.

We implemented a simple *monitor* which takes the discovered patterns and the difference parameters as inputs and outputs all those patterns also showing

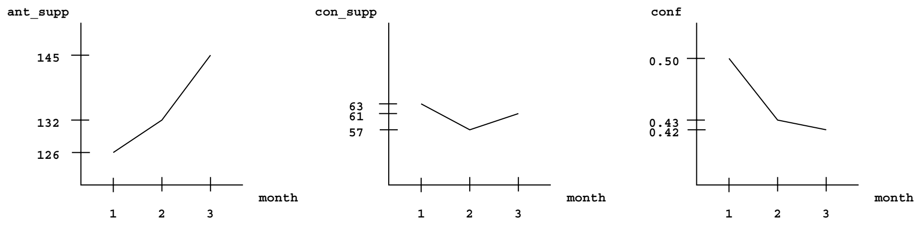


Fig. 3. A time series of a rules statistics.

the specified strength in their change. For this purpose, all patterns have been generated from the datasets. After importing them into the DBMS a series of SQL statements produces a set of patterns that meet the requirements given by the analyst. For example, if a rise of 20% in the support of the antecedent is considered interesting, the following SQL query would generate these patterns:

```
SELECT n.ant_page, n.ant_occ, n.con_page, n.con_occ,
    (n.ant_supp / (f.ant_supp - n.ant_supp)) AS delta
  FROM november n, february f
  WHERE n.ant_page = f.ant_page
  AND   n.ant_occ = f.ant_occ
  AND   n.con_page = f.con_page
  AND   n.con_occ = f.con_occ
  AND   abs(n.ant_supp / (f.ant_supp - n.ant_supp)) > 0.2
```

We computed a large set of patterns that showed interesting changes. However, it turned out that this approach only led to useful results if lower boundaries for support and confidence were also given. Moreover, there is a big overhead when generating all patterns from the datasets.

5 Conclusions and Future Work

In this work we have developed a framework which allows the management and the maintenance of mining results. We model the rules output by consecutive mining sessions as temporal objects, which may change in terms of statistics and in terms of contents. To monitor statistical changes of a rule, we treat rule statistics as time series, on which prediction and similarity searches can be performed using conventional techniques of time series analysis. Changes in a rules contents are harder to monitor. We are confronted with the fundamental question of whether a change in the contents of a rule should be observed as a mutation of the rule or as a new rule that has replaced the old one. As far as not given directly by the data, our framework delegates this decision to the analyst who can explicitly define which changes to a rule contents should be observed as rule "mutations", thus effectively supplying a unique identifier for rules across the mining sessions. In a case study we have shown the applicability of the proposed framework. Moreover, we have implemented a *monitor* which can observe interesting changes in these patterns.

The proposed work is only a first step in the direction of supporting the maintenance and monitoring of non-crisp knowledge that evolves across mining sessions. It should be extended in several ways. Firstly, we intend to combine our framework with a time series analysis tool and establish a complete environment for the monitoring of the statistic evolution of rules. Secondly, we will extend our approach to investigate also changes of a rules contents, especially when a global key is missing. In the case study we have also discovered patterns that disappeared from one mining session to another and re-appeared in the following mining session. To identify those rules without a global identifier is also challenging. Finally, the incorporation of a theorem prover that assesses new rules from the non-crisp mining results seems an indispensable component of a full-fledged knowledge base.

References

1. N. F. Ayan, A. U. Tansel, and E. Arkun. An Efficient Algorithm To Update Large Itemsets With Early Pruning. In *Proc. of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 287–291, San Diego, CA, USA, August 1999. ACM.
2. S. Baron and M. Spiliopoulou. Monitoring the Results of the KDD Process: An Overview of Pattern Evolution. In J. M. Meij, editor, *Converting Data into Knowledge*, chapter 5. The Netherlands Study Center for Technology Trends, Den Haag, Netherlands, to appear in Sep. 2001.
3. S. Chakrabarti, S. Sarawagi, and B. Dom. Mining Surprising Patterns Using Temporal Description Length. In *VLDB'98*, pages 606–617, New York, USA, Aug. 1998. Morgan Kaufmann.
4. X. Chen and I. Petrounias. Mining Temporal Features in Association Rules. In *Proceedings of the 3rd European Conference on Principles of Data Mining and Knowledge Discovery*, pages 295–300, Prague, Czech Republic, September 1999. Springer.
5. D. W. Cheung, S. D. Lee, and B. Kao. A General Incremental Technique for Maintaining Discovered Association Rules. In *DASFAA'97*, Melbourne, Australia, Apr. 1997.
6. D. W. Cheung, V. T. Ng, and B. W. Tam. Maintenance of Discovered Knowledge: A Case in Multi-Level Association Rules. In *KDD'96*, 1996.
7. M. Ester, H.-P. Kriegel, J. Sander, M. Wimmer, and X. Xu. Incremental Clustering for Mining in a Data Warehousing Environment. In *VLDB'98*, pages 323–333, New York, USA, August 1998. Morgan Kaufmann.
8. R. Feldman, Y. Aumann, A. Amir, and H. Mannila. Efficient Algorithms for Discovering Frequent Sets in Incremental Databases. In *DMKD'97*, Tucson, USA, Mai 1997.
9. V. Ganti, J. Gehrke, and R. Ramakrishnan. A Framework for Measuring Changes in Data Characteristics. In *Proceedings of the Eighteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 126–137, Philadelphia, USA, May 1999. ACM Press.
10. V. Ganti, J. Gehrke, and R. Ramakrishnan. DEMON: Mining and Monitoring Evolving Data. In *Proceedings of the 15th International Conference on Data Engineering*, pages 439–448, San Diego, USA, February 2000. IEEE Computer Society.
11. E. Omiecinski and A. Savasere. Efficient Mining of Association Rules in Large Databases. In *BNCOD'98*, pages 49–63, 1998.
12. M. Spiliopoulou. The Laborious Way from Data Mining to Web Mining. *Int. Journal of Comp. Sys., Sci. & Eng., Special Issue on "Semantics of the Web"*, 14:113–126, Mar. 1999.
13. M. Spiliopoulou and L. C. Faulstich. WUM: A Tool for Web Utilization Analysis. In *EDBT Workshop WebDB'98*, Valencia, Spain, Mar. 1998. Springer.
14. S. Thomas, S. Bodagala, K. Alsabti, and S. Ranka. An Efficient Algorithm for the Incremental Updation of Association Rules in Large Databases. In *KDD-97*, pages 263–266, Newport Beach, USA, Aug. 1997.
15. K. Wang. Discovering Patterns from Large and Dynamic Sequential Data. *Intelligent Information Systems*, 9:8–33, 1997.

Discovering Knowledge from Meteorological Databases: A Meteorological Aviation Forecast Study

Sérgio Viademonte and Frada Burstein

School of Information Management and Systems
Monash University, Australia
{sergio.viademonte;frada.burstein}@sims.monash.edu.au

Robert Dahni and Scott Williams

Bureau of Meteorology
Melbourne Victoria AUSTRALIA
{R.Dahni;S.Williams}@bom.gov.au

Abstract: In many application areas there are large historical data containing useful knowledge for decision support. However, this data taken in its raw form is usually of a poor quality. Thus it has very little value for the user-decision-maker if not adequately prepared. The Knowledge Discovery in Databases (KDD) is concerned with exploiting massive data sets in supporting use of historical data for decision-making. This paper describes an ongoing research project in the context of meteorological aviation forecasting, concerned with fog forecasting. The paper discusses the stages for performing knowledge discovery in the meteorological aviation-forecasting domain. The data used for research was taken from a real data set describing the aviation weather observations. The paper presents the data pre-processing stage, the discovered rules, achieved results and further directions of such research. We believe that this project[1] can serve as a model for in a wider KDD-based decision support problem.

1 - Introduction

Information about past decisions and their outcomes can help decision-makers identify potential successes and failures. The Knowledge Discovery in Databases (KDD) is concerned with supporting decision-making involving, normally massive, data sets. It is a multistage process that can be used to find useful knowledge in the form of patterns from the data describing past experience. The paper discusses the knowledge discovery stages, including data collection and analysis, feature selection, data cleaning and transformation. The paper also discusses the generated data models for data mining purposes and the used data mining algorithm. This project is being implemented in the context of aviation weather forecasting, specifically, fog phenomenon.

The production of meteorological forecasts begins with collection of weather observations, which describe the state of the atmosphere, e.g., amount of rainfall, information on the ground about air and wind-related measures, such as direction and

[1] **Acknowledgments:** We would like to thank the Australian Bureau of Meteorology, Victorian Regional Office for providing meteorological data and our industry collaborator, humanIT GmbH, Germany, for providing its data browser software InfoZoom, which assisted us in data pre-processing.

speed, temperature, wet-bulb and dew point temperature and sea-level pressure. Data available for predicting weather phenomena is usually highly diverse, high in volume and comes from different sources (Buchner at al 1998). This volume of data must be sorted and arranged into a comprehensible form before it is useful for predicting future weather patterns. Consequently, there is an overload of available data that makes KDD potentially useful. On the other hand, weather observations are collected and stored without necessary care for KDD purposes, lack of consistency and quality problems are common in such situations. It makes extensive pre-processing necessary.

Section 2 present the problem domain of the aviation weather forecasting, specifically in the area of Melbourne Tullemarine Airport; section 3 describes the data pre-processing stage; section 4 discusses the data-mining stage, presents the data-mining algorithm and data models. Sections 5 and 6 present some achieved results and the final conclusions.

2 – The aviation weather forecasting domain

The aviation weather forecasting is concerned with three main phenomena**:** *thunderstorm activity*, *low cloud* and *fog*. This research project is particularly focused on forecasting fog. Fog can be defined by visibility less than 1000 meters; and visibility between [1000 , 2000] meters (Auer, 1992).

Two different types of weather forecasts are made for Melbourne Airport. Terminal Aerodrome Forecasts (TAF) is issued every 6 hours for a forecasting period of 24 hours, containing data for: *time*, *location* and *weather conditions*. Every half an hour staff at the airport issue a Trend Type Forecast (TTF) appended to an observation of current weather. At Melbourne Airport the vast majority of poor weather conditions involve low cloud and fog, radar is of value for only a relatively small percentage of sub-minimum conditions. And the low cloud and fog does not only cover the airport but forms at a broader area, so the satellite pictures do not help to solve the problem. Access to the past situations and knowledge derived from them through KDD application can provide valuable source of improvement in forecasting rare events.

The data set for this study was generated from ADAM (Australian Data Archive for Meteorology) data repository. The data set, with weather observations from July 1970 until June 2000, has 49,901 data records and 17 attributes. These are the observations used when issuing an aviation meteorological prognosis bulletin. The data represents weather observations, for example, a particular day when fog was registered (Table 1).

Table 1. A reported observation of fog.

Case: Weather observations when Fog was observed on morning, February.
Wind speed 4.1 m/s
Dryer moderate to fresh westerly winds during the day before.
Dew Point dropping from 15 C in the morning to 5 C degrees – one day before
Easterly wind in the evening
Low cloud in the morning = 1 h.
Dew point temperature = 10C
Sea level pressure = 1022.0

2.1 Understanding the data

To select an appropriate data model concerning the problem at hand all the collected data must be well understood. The collected data was discussed and analyzed with domain experts. This section discusses the main characteristics about data interpretation.

Seasonal factor: attributes *Month*, *Year*, *Day* and *Hour* are date stamps and monotonic variables. The seasonal factor should be modeled monthly. The year and day attributes are not necessary for mining purposes, however they must be kept in the dataset, as they are necessary to calculate the previous afternoon dew point attribute, which is a new calculated field.

Fog identified attributes. There are two attributes that identify a particular occurrence of fog, "*Fog Type*" and "*Fog Occurrence*". When "*Fog Type*" has no value it means NO FOG event at that time. "*Fog Occurrence*" represents whether it is Fog Day (FD) or Local Fog Day (LFD); or is not a fog day at all. The distinction between local fog and fog day does not matter in terms of aviation weather forecasting. Therefore, the attribute "*Fog Occurrence*" is not necessary for the purposes of this study, as we focus on fog phenomenon occurrence in general.

"Hour" attribute. refers to a 3 hours period observation. The weather observations are recorded at a granularity of three hours. This granularity must be considered when making a prognostic, because the forecast prognosis issued in this project is based in 3 hours period observations.

Temperatures: The dew point and dry bulb are the local temperatures at the airport, taken when the event is observed.

Wind: The information about wind includes direction and speed. Both are taken at the surface at the airport, around 200 feet altitude. The wind direction attribute is expressed in degrees of compass points, from 0° to 360°, and the speed in meters/second.

All other information such as the *amount of clouds*, *sea level pressure*, *rainfall* and *visibility* is taken locally at the airport. Among them, visibility is highly important because the problem with dense fog is that it causes a lack of visibility.

3 - The data preprocessing stage

The importance of the preprocessing stage in the KDD process should not be underestimated. If the quality of the data is low or the problem is incorrectly formalized, the algorithms used for data mining will behave inefficiently or even produce incorrect results. As already mentioned, meteorological data is usually taken and stored without necessary care for KDD purposes. It makes extensive pre-processing necessary for achieving high overall results. There are many characteristics inherent to the data, such as sparsity, monotonicity, dimensionality, outliers, anachronisms and variables with empty or missing values, which present problems when preparing data for modeling (Pyle, 1999).

The first database used in this project had 75 attributes, some of them related with data quality control and codes describing observations. For example, the *dew point* observation had two associated attributes, one named *DWPT* and other named *DWPT_QUAL*, this last attributes indicates a data quality control information, which was not relevant for our data mining purposes. Many other observations, like *wind speed*, *wind direction* and *visibility* fall in the same category. In the first database, fog phenomenon was described through a Boolean type attribute (FOG_OCC) together with visibility attribute represented by codes. This leads to the following situation when identifying fog phenomenon:

- If Visibility is one of (12, 28, 42-49, 11, 40-41) then FOG_OCC = yes, otherwise FOG_OCC = no.

The following descriptions were obtained from data mining sessions

```
FOG = TYPE 40  96.1 % of the examples
Fog at a distance but no fog at the station during the past
hour, visibility greater than 1000m.


FOG = TYPE 12 2.0 % of the examples
Shallow Fog (below 2m), generally continuous
```

Instead of fog phenomenon identification, it identified fog types. Clearly, this was not the desirable situation. Our first experiments showed that the problem of fog forecasting would not be properly addressed without an intensive data preprocessing. The following section describes some data preprocessing for this study.

3.1 – Feature selection and feature construction

The features in the data set were mostly selected by the forecasters. However, some general analysis was still performed. The Y*ear* and *Day* attributes were not necessary for mining purposes. The attribute *Fog Occurrence* was removed, the project focuses on fog phenomenon occurrence regardless of its classification.

A derived attribute *previous afternoon dew point* was calculated based in the *date, hour* and the *dew point* and inserted in the data table. The forecasters recommended this information as very important for fog prognosis. For that reason *date* attribute were not removed from the database despite of not being used in any data mining session. A new attribute was inserted, *ID* auto number type. It is an identification attribute, which is necessary when importing the table into the data mining software.

3.2 – Missing and empty values

Handling missing and empty values is one of the most common problems in data preparation for data mining (Catlett, 1991; Pyle, 1999). Determining if any particular variable is empty rather than missing requires domain knowledge and is extremely difficult to be automatically detected. Missing and empty values can be removed or replaced. Other common problem of data analysis is related to low information density of a certain variable (*sparsity*). Normally, sparse variables have insignificant values and can be removed. However, there are cases when sparse values hold significant information. To remove or not a sparse variable is an arbitrary decision based on confidence levels (Pyle, 1999). Table 2 shows the amount of null values, which could be empty or missing, for each attribute in the dataset. *Rainfall* attribute shows a significant amount of null instances, 30.42 %. This percentage of null values could be enough to discard the attribute for data mining purposes; however, according to the experts the *Rainfall* is highly significant attribute and needs to be kept.

Table 2. Analyses of null values

Attributes	Amount of nulls values
Dry-bulb temperature	16 nulls
Dew point temperature	31 nulls
Previous dew point	572 nulls
Total cloud amount	25 nulls
Total low cloud amount	314 nulls
Mean sea level pressure	25 nulls
Past weather	93 nulls
Rainfall	15180 nulls
Present weather	81 nulls
Visibility	14 nulls
Wind direction	104 nulls
Wind speed	29 nulls

The attribute *Previous dew point* temperature also shows a significant amount of nulls instances, 11.46 %. However, doing a specific analysis with just *fog* cases, the attribute shows a smaller frequency of null values.

During the first analysis *Fog Type* attribute showed a high amount of null values, 98.07 %. Here a null value means neither missing, nor empty, but indicates that a particular weather observation is not a fog phenomenon. The *Fog Type* attribute was transformed to properly represent this information (see section 3.4 for description of data transformation).

Analyses of sparse attributes were individually made for each class, *fog* cases and *not fog* cases. The attributes without null values in the class were not considered in this analysis. Except of the *Rainfall* attribute, the others did not have significant amount of null values to be considered sparse; the "*Total cloud amount*" and "*Total low cloud amount*" have 1.27 % and 2.23 % of null values respectively, which is not significant enough to these attributes be considered sparse. The analysis of the *not fog class* has been done in a similar way; it resulted in just the *Rainfall* attribute classified as sparse.

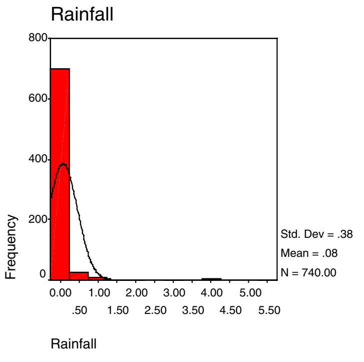
3.3 – Variability

Another important issue to consider is the data distribution, or variability. The possible patterns enfolded in a variable in some particular way are fundamental to discover the relationships among variables. Analysing the variability in the data set is important in order to verify the occurrence of constants, outliers and monotonic variables. The standard deviation was used to calculate the variability of numeric variables using SPSS Package.

The *Rainfall* attribute shows two problematic behaviors for data mining: sparsity and lack of variability. It has 198 nulls; representing 21.11 % of the total *fog class* instances and has 14.982 nulls; representing 30.60 % of the total *not fog class* instances. Figure 1 shows the *Rainfall* attribute frequency distribution for *fog* class:

Common approaches to handle this situation recommend removing the attribute, removing just the rows with null values, replacing the null values or transforming the attribute. According to domain expert, the null values in *Rainfall* attribute are considered zero values, for forecasting purposes. Also according to factual domain knowledge, a fog event is unlikely to happen during rainfall. However, transforming all the null values into zero values leads to the problem of lack of variability, as, the null values will represent 812 instances out of 938 (86,57 %). In this particular case the *Rainfall* attribute could be considered a zero value constant. Section. 3.4 describes the approach used to handle this problem.

Another problem that was noted is the fog observations with high visibility values, near and over 30 kilometers. The *visibility* variable in that case acts as an outlier variable. (see Figure 2).



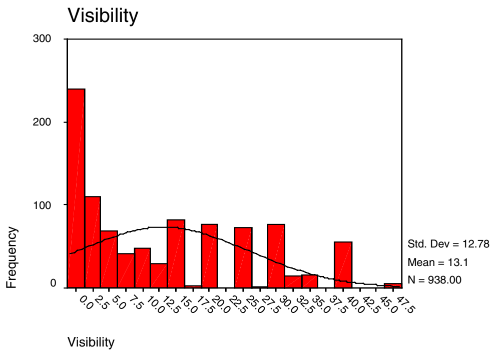


Fig. 1. *Rainfall* histogram for *fog* class **Fig. 2.** *Visibility* histogram for *fog* class

The *visibility* axis of the histogram shows the cluster with higher values at the right handside. Values over 30 kilometers can be observed, while small values are expected. Fog is mainly characterized by low visibility (<1000 meters). The occurrence of fog with visibilities around 30 kilometers is an impossible situation. According to the domain experts those particular cases refer to *fog patches*, i.e. there are some patches of fog in the area, however the overall visibility is mostly fine. Therefore, the high visibility values were kept in the data set without further modifications.

Other problem was with the *Wind Direction* attribute, which is specified relative to the true geographic north, and it is the direction from which the wind is blowing. The *Wind Direction* is numerically represented in the BOM source data set in degrees of compass points. The *Wind Direction* has a high frequency of zero values and its neighborhood and a high frequency of 360 degrees. The problem here is that zero values and 360 values represent the same compass direction. However, zero value does not always mean the same as a 360-value degree. Next section describes how this problem was handled.

3.4 – Data transformation

Several data transformations were performed with Bureau original data set. For example, *Fog Type* attribute has 3 possible values assigned: "*F*" when it refers to a fog event, "*LF*" when is "Local Fog" and null when the event is not fog; see section 2 for detailed discussion. This study is concerned with occurrence of fog phenomenon regardless of whether it is a local fog or not. For this reason all the *Fog Type* instances with "*LF*" value were transformed in "*F*" value, meaning a *fog* case. All the instances of *Fog Type* with null value were assigned "*NF*" values, meaning *not fog* case.

The attributes *PastWeather* and *PresentWeather* were transformed from numeric type to non-numeric type, text size 4. These attributes are qualitative (categorical) attributes; they indicate weather codes.

Some transformations of *Rainfall* attribute were needed. Rainfall is measured for the 3 hours period ending at the nominated time. The rainfall volume is initially measured in millimeters and presented to the forecasters; who express their evaluation in codes expressing ranges of millimeters. This procedure makes sense according to the nature of the forecasting task, as it is almost impossible to differentiate precise measurements of rainfall, like 0.3 millimeters and 0.2 millimeters. The numerical values were transformed into categorical codes, which express ranges of rainfall. The instances of null rainfall will be classified into code 0, no rain. To implement this transformation a new attribute was inserted, Rainfall_Range text attribute of size 4. A procedure was implemented to calculate the Rainfall_Range attribute corresponding to Rainfall attribute values.

The third problem regarding data modeling is the *Wind Direction*. It is a measure taken by instruments and it is numerically represented in degrees. However, the forecasters do not use detailed numerical measurements when reporting a forecast bulletin but a categorical representation. A categorical description of compass point is used instead, being N for North, S for South and so on. The wind direction representation used in this research project is 16 compass points, also called VRB, used by the Bureau. For example, lets consider the compass point 22,1 degrees. Practically, in that case the forecasters assume the compass point NNE, which is the point 22.5 degree. In that case the wind direction is said to be NNE instead of 22.1, as for forecasting purposes the distinction among 22.1, 22.2 and 22.5 degrees is not significant; again a tolerance for imprecision can be observed. It is assumed that each value in degrees belongs to the closest compass point, therefore the middle point between each two compass points was chosen as the boundary point between them, being the middle point itself belonging to the next upper compass point.

To implement the transformation of *WindDirection* attribute from degrees to compass points a new attribute was inserted into the data set, the *WindCompas,* text attribute size

8. The *not fog class* data set initially had 48.963 instances. After all data transformations and after the nulls instances were removed, the resulted *not fog class* data set has 47,995 instances, and the *fog class* data set has 938 instances.

4 – The data-mining stage

Aviation weather forecasting can be studied as a classification problem, where the weather phenomena are the established patterns for classification, e.g., *fog*, *severe thunderstorms* and *normal weather condition*. The weather observations are the data under evaluation. Several weather observations and their possible combinations could be appraised, such as *wind speed*, *wind direction*, *dew point*, etc, when determining the occurrence of a certain weather pattern. Our modeling uses fifteen attribute values of weather observations and two pre-defined classes, *fog* and *not fog*. For combinations of order three, the cluster size is 848046 possible combinations; for combinations of order five the cluster size increased to 1218218079 combinations.[2]
After data pre-processing, the next stage is to select a set of data models for data mining, it means, to properly choose the sets of data for training (mining), test and evaluation. Data mining technology was applied to the set of cases stored in a case base (or *mining data set*) to find "chunks" of domain knowledge. Next sections discuss the generated data models and present a brief description of the used data-mining algorithm.

4.1 – Data models

Fog and *Not Fog* classes have a heterogeneous distribution making a homogeneous analysis difficult. The dataset has 938 instances of *fog* class and 47,995 instances of *not fog* class. Because of that, it was decided to split the dataset into two, one data set with *fog cases* and the other with *not fog cases*. Each class was split into *training* (mining), *testing* and *evaluation* data sets. It has been recognized that the data set must be large enough in order to capture sufficient information about each variable and the interactions among them (Mohammed,et al1996; Provost&Kolluri, 1999; Pyle, 1999).

Normally the whole dataset is randomly divided into about 80% for training purposes and 20% for testing or evaluation. Several exploratory studies and experiments handling data sampling are reported in specific literature about knowledge discovery, knowledge engineering and machine learning; see (Berry&Linoff, 2000; Buchner et al., 1998; Chen et al 1998; Hruschka&Ebecken, 1998; Siegler&Steurer, 1998) for illustrating data sampling approaches. The sampling approach used in this study was the stratified sampling. This approach is suitable when the classes in the population are heterogeneously distributed (Catlett, 1991).

For *fog class* the whole population was used being 85% of the population randomly selected for mining data set, with 807 rows of data. In addition, 7% of the population was randomly selected for testing and for evaluation, each one, with 63 and 68 rows of data respectively.

For *not fog class*, two samples were generated from the whole population, one sample being 10% of the whole population, named *Model1,* with 4,763 instances. The second sample named *Model2*, 20% of the population with 9,572 instances. From *Model1*, three data sets were randomly generated; a mining data set (60% of the sample), called *Model1-60*, an evaluation data set and a test data set, both being 10% out of the *Model1*. In addition, a second mining data set was generated, being 80% of the initial sample (*Model1*) and new evaluation data set and test data set were also generated, both of them being 10% out of the *Model1* sample. Those data sets belong to *Model1-80*.

[2] Combinations of "N" objects, "R" at a time is given by: $C_r^n = n! / r! (n-r)!$, where '!' indicates factorial operation.

Similar approach was used with *Model2*. A mining data set being 60% of the sample, an evaluation data set and a test data set, both of them being 10% were generated from the *Model2* sample. Those data sets are named *Model2-60*. Table 3 illustrates the generated data models for *not fog* class.

Table 3. *Not fog* class data models

Data Model	Model 1-60	Model 1-80	Model 2-60
Population size (N)	47.995	47.995	47.995
Sample size (n)	4.763 (10% / N)	4.763 (10% / N)	9.572 (20% / N)
Mining set	2.836 (60% / n)	3.869 (80% / n)	5.699 (60% / n)
Evaluation set	490 (10% / n)	438 (10% / n)	949 (10% / n)
Test set	500 (10% / n)	456 (10% / n)	992 (10% / n)

The final data sets for mining were obtained by joining the *fog class* mining set with each *not fog class* mining set, therefore three models of mining set were generated. Table 4 shows the mining set models:

Table 4. Mining data models

	Mining Model 1-60	Mining Model 1-80	Mining Model 2-60
Mining sets	Fog mining Model + Not fog Model 1-60	Fog mining Model + Not fog Model 1-80	Fog mining Model + Not fog Model 2-60
Rows in fog class	807	807	807
Rows in not fog class	2.836	3.869	5.699
Total number of rows	3.643	4.676	6.506

The samples were generated using SPSS random sampling generation mechanism. Analyses of variability were performed between the samples and the population to check that the samples had captured the population variability. The evaluation data set will be used to handle to problem of overfitting in the neural network, and the test data set will be used to verify the level of learning and accuracy.

4.2 – Associative Rule Generator Algorithm

We characterized the aviation weather forecasting problem as a classification problem, consequently an approach to handle classification problems was chosen. The used approach is an association rules algorithm (Agrawal, et al 1993; Mohammed et al., 1996.). An association rule is an expression X →Y, where X and Y are sets of predicates; where X is a precondition of the rule in disjunctive normal form and Y - the target post condition. The *Combinatorial Rule Model* (CRM) (Beckenkamp et al 1998) was the algorithm used in the experiments so far in this project. CRM is a data mining oriented version of the *Combinatorial Neural Model* (CNM) algorithm (Beckenkamp et al., 1998; Machado, Barbosa&Neves, 1998). The CNM network topology is a feed-forward network with a topology of 3 layers.

The input layer of a CNM network represents the set of evidences about the examples. That is, each input is in the [0,1] interval, indicating the pertinence of the training example to a certain concept. The combinatorial layer is automatically generated: a neuron is added to this layer for each possible combination of evidences, from order 1 to a maximum order, given by the user. The output layer corresponds to the possible classes an example could belong (Machado et al., 1998). Combinatorial neurons behave as conjunctions of evidences that lead to a certain class.

The use of CRM algorithm is a preliminary study and does not imply any particular preference. Others data mining algorithms are under evaluation to be used in this research project. For example the ARMiner algorithm and some tree induction algorithms, such as C4.5 (Quinlan, 1993) will be further applied.

5 – Results and analysis

Experiments with CRM algorithm were carried out with the described data models. The meteorological database has been used to find patterns describing the occurrence of dense fog at Tullemarine airport, Melbourne, Australia. The output attribute, *Fog Type* was discretised into classes "*F*" for *Fog* and "*NF*" for *Not Fog*.

In an exploratory approach, rules with 70%, 80% and 90% of confidence degree levels were generated and have been appraised to select the most relevant ones. All the experiments were done using a minimum rule support of 10%. The support represents the ratio of the number of records in the database for which the rule is true to the total number of records. The confidence degree represents the conditional probability of the consequent given the antecedent (Buchner et al., 1998; Piatetsky et al 1991). Table 5 shows the amount of generated rules for each data models with the confidence level accordingly.

Table 5. Amount of generated rules for each data model

Data Model	Rule Confidence Degree					
	70%			80%		
	Fog	Not Fog	Total	Fog	Not Fog	Total
Model 1-60	108	134	242	74	130	204
Model 1-80	101	126	227	61	126	187
Model 2-60	83	189	272	45	189	234

For the confidence level of 90% too few rules were obtained; with a maximum amount of 13 rules for *fog class* when using data model *Model1-80*. This amount of rules is unlikely to be enough for a satisfactory description of the *fog* phenomenon. Experiments with the confidence factor of 70% and 80% resulted in several rules some of which are very trivial for decision support. They may be discarded during the rule verification process.

Example of discovered rule from *Model1-60*, *Fog class*, are showed below:

```
Rule 1 for Fog Class:
If TotalLowCloud > 7
      And Rainfall  = 0
      And Visibility <= 24
      And Wind Speed <= 1
Then Fog Type = F (Confidence: 94.74%, Number of cases: 108,
Support: 13.38%,)
```

The rules need to be appraised by the forecasters for their accuracy and usefulness. However, the outcomes show the feasibility of knowledge discovery technology in aviation weather forecasting. New experiments will be performed, selecting different weather observations.

6 – Conclusion and Comments

This paper presents a meteorological knowledge discovery experiment; it addresses the application of KDD for industrial problems. Data selection task, the prep-processing stage; the data-mining algorithm and data model were also discussed. The applicability of this technology to support aviation weather forecasting has been shown.

The results obtained so far were presented to the forecasters from the Bureau and assessed as potentially useful in supporting their tasks. Also, if compared with the statistical methods currently used for severe weather forecasting, the generated set of rules showed a higher level of generalizability of relations stored in a case base.

Next steps in this study will deal with rule evaluation and verification. Those selected rules will be stored in a knowledge base and powered by an expert system inference engine for decision support. Fuzzy logic is potentially useful for representing semantic variables such as *Wind Speed* and *Wind Direction* to avoid excessively simplifications. This approach will be further evaluated. The use of different data mining algorithms is under evaluation, and a comparative study is planned.

References

Auer, A. H. J. (1992). *Guidelines for Forecasting Fog. Part 1: Theoretical Aspects*: Meteorological Service of NZ.

Agrawal, R., Imielinski T.&Swami, A. (1993). Mining association rules between sets of items in large databases. In *Proceedings of Conference Management of Data.*

Beckenkamp, F., Pree, W.&Feldens, M. A. (1998). Optimizations of the Combinatorial Neural Model. In *Proceedings of 5th Brazilian Symposium on Neural Networks (SBRN'98)*, Belo Horizonte, Brazil.

Buchner, A. G., Chan, J. C. L., Hung, S. L.&Hughes, J. G. (1998). A meteorological knowledge-discovery environment. *Knowledge Discovery and Data Mining.* (pp.204-226).

Catlett, J. (1991). *Megainduction: Machine learning on very large databases.* UTS, Australia.

Chen, F., Figlewski, S., Weigend, A. S.,&Waterhouse, S. R. (1998). Modeling financial data using clustering and tree-based approaches. In *Proceedings of International Conference on Data Mining*, (pp.35-51). Rio de Janeiro, Brazil: WIT Press.

Fayyad, U. M., Mannila, H.&Ramakrishman, R. (1997). *Data Mining and Knowledge Discovery* (Vol.3). Boston: Kluwer.

Gottgtroy, M. P. B., Rodrigues, M. J. N.&Sousa, M. T. G. (1998). Data mining agents. In *Proc. of Intern.Conf on Data Mining* (171-182). RiodeJaneiro, Brazil:WIT Press,Southampton, UK.

Howard, C. M.&Rayward-Smith, V. J. (1998). Discovering Knowledge from low-quality meteorological databases., *Knowledge Discovery and Data Mining.* (pp.180-202.).

Hruschka, E.&Ebecken, N. (1998). Rule Extraction from Neural Networks in Data Mining Applications. In *Proc. of International Conference on Data Mining*, (pp.303-314). RiodeJaneiro, Brazil: WIT Press, UK.

Keith, R. (1991). *Results And Recommendations Arising From An Investigation Into Forecasting Problems At Melbourne Airport.* (MeteorologicalNote 195). Townsville: Bureau of Meteorology, Meteorological Office.

Machado, R. J., Barbosa, V. C.&Neves, P. A. (1998). Learning in the Combinatorial Neural Model. *IEEE Transactions on Neural Networks, 9.* September 1998

Mohammed, J. Z., Parthasarathy S., Li W.&Ogihara, M. (1996). *Evaluation of Sampling for Data Mining of Association Rules.* (Tech.Rep. 617). Rochester, New York: The University of Rochester, Comp. Sci.Dept.

Piatetsky-Shapiro, G.,&Frawley, W. (1991). *Knowledge Discovery in Databases.*: MIT Press.

Provost, F.,&Kolluri, V. (1999, June, 1999). A Survey of Methods for Scaling Up Inductive Algorithms. *Data Mining and Knowledge Discovery, Volume 3*, 131-169.

Pyle, D. (1999). *Data Preparation for Data Mining*. San Francisco, USA:Morgan Kaufmann Publishers, Inc.

Quinlan, J. R. (1993). C4.5: Programs for Machine Learning. CA: Morgan Kaufmann.

Siegler, W.&Steurer, E. (1998). Forecasting of the German stock index DAX with neural networks: Using daily data for experiments with input variable reduction and a modified error function. In *Proc. of International Conference on Data Mining.* (pp.289-301). RiodeJaneiro, Brazil:WIT Press, Southampton, UK.

Enhancing the *Apriori* Algorithm for Frequent Set Counting

Salvatore Orlando[2], P. Palmerini[1,2], and Raffaele Perego[1]

[1] Istituto CNUCE, Consiglio Nazionale delle Ricerche (CNR), Pisa, Italy
[2] Dipartimento di Informatica, Università Ca' Foscari di Venezia, Italy

Abstract. In this paper we propose **DCP**, a new algorithm for solving the Frequent Set Counting problem, which enhances *Apriori*. Our goal was to optimize the initial iterations of *Apriori*, i.e. the most time consuming ones when datasets characterized by short or medium length frequent patterns are considered. The main improvements regard the use of an innovative method for storing candidate set of items and counting their support, and the exploitation of effective pruning techniques which significantly reduce the size of the dataset as execution progresses.

1 Introduction

The *Frequent Set Counting* (FSC) [1,2,3,4,6,11,12,13,15,17,19] problem has been extensively studied as a method of unsupervised Data Mining [8,9,16] for discovering all the subsets of items (or attributes) that frequently occur in the transactions of a given database. Knowledge of the frequent sets is generally used to extract *Association Rules* stating how a set of items (itemset) influences the presence of another itemset in the transaction database. In this paper we focus on the FSC problem, which is the most time-consuming phase of the Association Mining process. An itemset is *frequent* if it appears in at least $s\%$ of all the n transactions of the database $\mathcal{D}$, where s is the *support* threshold. When $\mathcal{D}$ and the number of items included in the transactions are huge, and we are looking for itemsets with small support, the number of frequent itemsets becomes very large, and the FSC problem very expensive to solve, both in time and space. Although a number of other solutions have been proposed, *Apriori* [4] is still the most commonly recognized reference to evaluate FSC algorithm performances. *Apriori* iteratively searches frequent itemsets: at each iteration k, F_k, it identify the set of all the itemsets of k items (k-itemsets) that are frequent. In order to generate F_k, a *candidate* set C_k of potentially frequent itemsets is first built. By construction, C_k is a superset of F_k. Thus, to discover frequent k-itemsets, the support of all candidate sets is computed by scanning the entire transaction database $\mathcal{D}$. All the candidates with minimum support are then included in F_k, and the next iteration is started. The algorithm terminates when F_k becomes empty, i.e. when no frequent set of k or more items is present in the database. It is worth considering that the computational cost of the k-th iteration of *Apriori* strictly depends on both the cardinality of C_k and the size of $\mathcal{D}$. In fact, the number of possible candidates is, in principle, exponential in the number m of items appearing in the various transactions of $\mathcal{D}$. *Apriori* considerably reduces the number of candidate sets on the basis of a simple but very effective observation: a k-itemset can be frequent only if all its subsets of $k-1$ items are frequent. C_k is thus built at each iteration as the set of all k-itemsets whose subsets of $k-1$ items are all included in F_{k-1}. Conversely, k-itemsets that contain at least one infrequent $(k-1)$-itemset are not included in C_k. This level-wise behavior characterizes a whole family of algorithms for FSC, as for example the Direct Hash with Pruning algorithm (DHP) [15]. An important algorithmic problem addressed by these algorithms is to efficiently count the support of the candidate itemsets. During iteration k, all the k-subsets of each transaction $t \in \mathcal{D}$ must be determined and their presence in C_k be checked. To reduce the complexity of this phase, *Apriori* stores the various candidate itemsets in the leaves of a *hash-tree*, while suitable hash tables are placed in the internal nodes of the tree to direct the search of k-itemsets within C_k. The performance, however, only improves if the hash-tree splits C_k into several small disjointed partitions stored in the leaves of the tree. Unfortunately this does not happen for small values of k since the depth of the tree and thus the number of its leaves depends on k.

DCP (Direct Count of candidates & Prune transactions), the new algorithm proposed in this paper, enhances *Apriori* by exploiting an innovative method for storing candidate itemsets and counting their support. The method is a generalization of the *Direct Count* technique used for counting the support of unary itemsets, and allows the cost of the initial iterations of the algorithm to be reduced considerably, both in time and space. Moreover, to reduce both computation and I/O cost, a pruned dataset $\mathcal{D}_{k+1}$ is written to the disk at each iteration k of the algorithm, and employed at the next iteration. **DCP** adopts simple and effective heuristics for pruning the dataset, without using the complex hash filter used by DHP [15]. **DCP** does not address the issues arising in databases from which very long patterns can be mined [1,6]. In this case we have an explosion of the candidate number, since all the 2^l subsets of each long maximal frequent itemset of length l have to be produced and counted. More importantly, in these problems the supports of short frequent k-itemsets are usually very large, thus making the counting process very expensive. To validate our algorithm, we conducted in-depth experimental evaluations by taking into account not only execution times, but also virtual memory usage, I/O activity and their effects on the elapsed time. When possible, locality of data and pointer dereferencing were optimized due to their importance with respect to the recent developments in computer architectures. The experimental results showed that our new algorithm, **DCP**, outperforms the others. Our test bed was a Pentium-based Linux workstation, while the datasets used for tests were generated synthetically.

The paper is organized as follows. Section 2 introduces **DCP**, and discusses its features in depth. Section 3 reports the promising results obtained, while Section 4 reviews some of the most recent results in the FSC field. Finally, Section 5 draws some conclusions.

2 The DCP Algorithm

As with the other algorithms of the *Apriori* class, **DCP** uses a *level-wise, counting-based* approach, according to which at each iteration k all the transactions are counted against a set of candidate k-itemsets. The database $\mathcal{D}$ is horizontally stored, i.e. each record corresponds to a variable length transaction containing a set of items represented by integer identifiers. We assume that item identifiers are stored in sorted order in both transactions and itemsets, and that sets C_k and F_k, $F_k \subseteq C_k$, are both stored as lexicographically ordered vectors of k-itemsets.

At each iteration k, **DCP** builds C_k on the basis of F_{k-1}. In this construction we exploit the lexicographic order of F_{k-1} to find pairs of $(k-1)$-itemsets sharing a common $(k-2)$-prefix. Due to this order, the various pairs occur in close positions within the vector storing F_{k-1}. The union of each pair is a k-itemset that becomes a candidate $c \in C_k$ only if all its subsets are included in F_{k-1}. Also in this case we can exploit the lexicographic order of F_{k-1}, thus checking whether all the subsets of c are included in F_{k-1} in logarithmic time.

The main enhancements introduced by **DCP** regard the exploitation of database pruning techniques, and the use of an innovative method for storing candidate itemsets and counting their support:

- A pruned (and smaller) dataset $\mathcal{D}_{k+1}$ is written to the disk at each iteration k, and employed at the next iteration. Pruning clearly reduces I/O activity but its main benefit is the reduced cost of the subset counting phase due to the reduced number and size of transactions.
- **DCP** does not use a hash tree for counting frequent sets. Instead it is based on directly accessible data structures, thus enhancing spatial locality and avoiding complex and expensive pointer dereferencing.

2.1 Pruning the Dataset

Two different pruning techniques are exploited. *Dataset global pruning* which transforms a generic transaction t, read from $\mathcal{D}_k$ into a pruned transaction $\hat{t}$, and *Dataset local pruning* which further prunes the transaction, and transforms $\hat{t}$ into $\ddot{t}$ before writing it to $\mathcal{D}_{k+1}$. While the former technique is original, the latter has already been adopted by DHP [15].

Dataset global pruning. The technique is based on the following argument: t may contain a frequent k-itemset I only if all the $(k-1)$-subsets of I belong to F_{k-1}. Since searching F_{k-1} for all the $(k-1)$-subsets of any $I \subseteq t$ may be very expensive, a simpler heuristic technique, whose pruning effect is smaller, was adopted. In this regard, note that the $(k-1)$-subsets of a given k-itemset $I \subseteq t$ are exactly k, but each item belonging to t should only appear in $k-1$ of these k itemsets. Therefore, we derive a necessary (but weaker) condition to keep a given item in t: *item t_i is retained in $\hat{t}$ if it appears in at least $k-1$ frequent itemsets of F_{k-1}*. To check the condition above, we simply use a *global* vector $G_{k-1}[\ ]$

that is updated on the basis of F_{k-1}. Each counter of $G_{k-1}[\ ]$ is associated with one of the m items of $\mathcal{D}_k$. For each frequent $(k-1)$-itemset belonging to F_{k-1}, the global counters associated with the various items appearing in the itemset are incremented. After F_{k-1} has been scanned, $G_{k-1}[j] = x$ means that item j appears in x frequent itemsets of F_{k-1}. Counters $G_{k-1}[\ ]$ are thus used at iteration k as follows. An item $t_i \in t$ is copied to the pruned transaction $\hat{t}$ only if $G_{k-1}[t_i] \geq k-1$. Moreover, if $|\hat{t}| < k$, the transaction is skipped, because it cannot possibly contain any frequent k-itemset.

Dataset local pruning. The *Dataset local pruning* technique is applied to each transaction $\hat{t}$ during subset counting. The arguments this pruning technique is based on, are similar to those of its global counterpart. Transaction $\hat{t}$ may contain a frequent $(k+1)$-itemset I only if all the k-subsets of I belong to F_k. Unfortunately, F_k is not yet known when our *Dataset local pruning* technique should be applied. However, since C_k is a superset of F_k, we can check whether all the k-subsets of any $(k+1)$-itemset $I \subseteq \hat{t}$ belong to C_k. This check could be made *locally* during subset counting of transaction $\hat{t}$.

Note that to implement the check above we should have to maintain, for each transaction $\hat{t}$, information about the inclusion of all the k-subsets of $\hat{t}$ in C_k. Since storing this information may be expensive, we adopted the simpler technique already proposed in [15], whose pruning effect is however smaller: *item $\hat{t}_i$ is retained in $\ddot{t}$ if it appears in at least k candidate itemsets of C_k.*

2.2 Direct Count of Frequent $k-$Itemsets

On databases characterized by short or medium length patterns, most of the execution time of *Apriori* is spent on the first iterations, when the smallest frequent itemsets are searched for. While *Apriori* uses an effective direct count technique for $k = 1$, the hash-tree data structure used for the other iterations, is not efficient for small values of k. For example, for $k = 2$ or 3, C_k is usually very large, and the hash tree used by *Apriori* splits it into only a few partitions, since the depth of the hash tree depends on k. Based on this observation, for $k \geq 2$ we used a *Direct Count* technique which is based on a generalization of the technique exploited for $k = 1$. The technique is different for $k = 2$ and for $k > 2$, so we will illustrate the two cases separately.

Counting frequent 2-itemsets. A trivial direct method for counting 2-itemsets can simply exploit a matrix of m^2 counters, where only the counters appearing in the upper triangular part of the matrix will actually be incremented [19]. Unfortunately, for large values of m, this simple technique may waste a lot of memory. In fact, since $|F_1|$ is usually less than m, and $C_2 = F_1 \times F_1$, we have that $|C_2| = \binom{|F_1|}{2} \ll m^2$. Before detailing the technique, note that at each iteration k we can simply identify M_k, the set that only contains the significant items that have not been pruned by the *Dataset global pruning* technique at iteration k. Let $\overline{m}_k = |M_k|$, where $\overline{m}_k < m$. In particular, for $k = 2$ we have that $M_2 = F_1$, so that $\overline{m}_2 = |F_1|$. Our technique for counting frequent 2-itemsets is thus based on the adoption of vector COUNTS[ ], which contains $|C_2| = \binom{\overline{m}_2}{2} = \binom{|F_1|}{2}$ counters. The counters are used to accumulate the frequencies of all the possible itemsets in C_2 in order to obtain F_2. It is possible to devise a perfect hash function to directly access the counters in COUNTS[ ]. Let $\mathcal{T}_2$ be a strictly monotonous increasing function mapping M_2 to $\{1, \ldots, \overline{m}_2\}$. The entry of COUNTS[ ] corresponding to a generic candidate 2-itemset $c = \{c_1, c_2\}$ can be accessed *directly* by means of the following order preserving, minimal perfect hash function:

$$\Delta_2(c_1, c_2) = \mathcal{F}_2^{\overline{m}_2}(x_1, x_2) = \sum_{i=1}^{x_1-1} (\overline{m}_2 - i) + (x_2 - x_1) = \overline{m}_2(x_1 - 1) - \frac{x_1(x_1-1)}{2} + x_2 - x_1, \quad (1)$$

where $x_1 = \mathcal{T}_2(c_1)$ and $x_2 = \mathcal{T}_2(c_2)$. Equation (1) can easily be derived by considering how the counters associated with the various 2-itemsets are stored in vector COUNTS[ ]. We assume, in fact, that the counters relative to the various pairs $\{1, x_2\}$, $2 \leq x_2 \leq \overline{m}_2$ are stored in the first $(\overline{m}_2 - 1)$ positions of vector COUNTS, while the counters corresponding to the various pairs $\{2, x_2\}$, $3 \leq x_2 \leq \overline{m}_2$, are stored in the next $(\overline{m}_2 - 2)$ positions, and so on. Moreover, the pair of counters relative to $\{x_1, x_2\}$ and $\{x_1, x_2+1\}$, where $1 \leq x_1 < x_2 \leq \overline{m}_2 - 1$, are stored in contiguous positions of COUNTS[ ].

Counting frequent k-itemsets. The technique above cannot be generalized to count the frequencies of k-itemsets when $k > 2$. In fact, although $\overline{m}_k$ decreases with k, the amount of memory needed to store $\binom{\overline{m}_k}{k}$ counters might be huge, since in general $\binom{\overline{m}_k}{k} \gg |C_k|$. Before detailing the technique exploited by **DCP** for $k > 2$, remember that, at step k, for every transaction t, we have to check whether any of its $\binom{|t|}{k}$ k-subsets belong to C_k. Adopting a naive approach, one could generate *all* the possible k-subsets of t and check each of them against *all* the candidates in C_k. The hash tree used by *Apriori* is aimed at limiting this check to only a subset of all the candidates. A *prefix tree* is another data structure that can be used for the same purpose [13]. In **DCP** we adopted a limited and directly accessible *prefix tree* to select subsets of candidates sharing a given prefix, the first two items of the k-itemset. Note that, since the various k-itemsets of C_k are stored in a vector in lexicographic order, all the candidates sharing a common 2-item prefix are stored in a *contiguous section* of this vector. To efficiently implement our *prefix tree*, a directly accessible vector PREFIX$_k$[ ] of size $\binom{\overline{m}_k}{2}$ is thus allocated. Each location in PREFIX$_k$[ ] is associated with a distinct 2-item prefix, and contains the pointer to the first candidate in C_k characterized by the prefix. More specifically, PREFIX$_k[\Delta_k(c_1, c_2)]$ contains the starting position in C_k of the segment of candidates whose prefix is $\{c_1, c_2\}$. As for the case $k = 2$, in order to specify $\Delta_k(c_1, c_2)$, we need to exploit a renumbering function $\mathcal{T}_k : M_k \rightarrow \{1, \ldots, \overline{m}_k\}$. $\Delta_k(c_1, c_2)$ can be thus defined as $\Delta_k(c_1, c_2) = \mathcal{F}_2^{\overline{m}_k}(x_1, x_2)$, where $x_1 = \mathcal{T}_k(c_1)$ and $x_2 = \mathcal{T}_k(c_2)$, while the hash function $\mathcal{F}_2^{\overline{m}_k}$ is defined by Equation (1). **DCP** exploits $PREFIX_k$[ ] as follows. In order to count the support of the candidates in C_k, we select all the possible prefixes of length 2 of the various k-subsets of each transaction $t = \{t_1, \ldots, t_{|t|}\}$. Since items within transactions are ordered, once a prefix $\{t_{i_1}, t_{i_2}\}$, $t_{i_1} < t_{i_2}$, is selected, the possible completions of this prefix needed to build a k-subsets of t can only be found in $\{t_{i_2+1}, t_{i_2+2}, \ldots, t_{|t|}\}$. The contiguous section of C_k which must be visited is thus delimited by both $\text{PREFIX}_k[\Delta_k(t_{i_1}, t_{i_2})]$ and $\text{PREFIX}_k[\Delta_k(t_{i_1}, t_{i_2}) + 1]$. Note that this counting technique exploits high spatial locality. Moreover, we highly optimized the code which checks whether each candidate itemset, selected through the prefix tree above, is included or not in $t = \{t_1, \ldots, t_{|t|}\}$. Our technique requires at most k comparisons. The algorithmic trick used is based on the knowledge of the number and the range of all the possible items appearing in each transaction t and in each candidate k-itemset c. In fact, this allows us to build a vector $POS[1 \ldots m]$, storing information about which items actually appear in t. More specifically, for each item t_i of t, $POS[t_i]$ stores its position in t, zero otherwise. The possible positions thus range from 1 to $|t|$. Therefore, given a candidate $c = \{c_1, \ldots, c_k\}$, c is not included in t if there exists at least one item c_i such that $POS[c_i] = 0$. Since both c and t are ordered, we can deduce that c is not a subset of t without checking all the items occurring in c. In particular, given a candidate $c = \{c_1, \ldots, c_k\}$ to be checked against t, we can derive that $c \nsubseteq t$, even if c_i actually appears in t, i.e. $POS[c_i] \not\models 0$. Suppose that the position of c_i within t, i.e. $POS[c_i]$, is such that $(|t| - POS[c_i]) \; < \; (k - i)$. If the disequation above holds, then $c \nsubseteq t$ because c contains other $(k - i)$ items greater than c_i, but such items in t are only $(|t| - POS[c_i]) \; < \; (k - i)$.

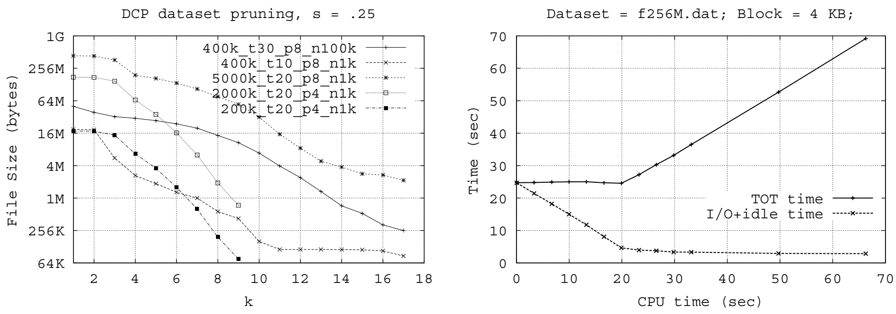


Fig. 1. DCP pruning efficacy, and I/O cost.

3 Performance Evaluation

We conducted several experiments to compare **DCP** with *Apriori*, DHP, and *Apriori*$_{DP}$, an implementation of *Apriori* which exploits our dataset pruning technique. The test bed architecture used was a 450MHz Pentium III Linux workstation with 512MB RAM and an Ultra2 SCSI disk, while the datasets were generated synthetically by using a commonly adopted dataset generator [4].

The plot reported in the left hand side of Figure 1 quantifies the effectiveness of **DCP** pruning. For $s = 0.25\%$ and various datasets, we plotted the size of $\mathcal{D}_{k+1}$

as a function of the iteration index k. As it can be seen, the reduction in the size of the processed dataset is valuable even with this low support threshold. Although our technique is slightly less effective than the DHP one in early iterations [14], it is much more efficient both in time and space from a computational point of view.

Dataset pruning has important implications on performances. Even if the size of the transaction database is initially huge, we can forecast that after few iterations the pruned dataset may entirely fit into the main memory. If this is the case, the original out-of-core algorithm could become *in-core*. Note that this transformation is automatically achieved by modern OSs that exploit the main memory left unused by the kernel and the processes as a *buffer cache* for virtualizing accesses to block devices such as disks [5]. Moreover, since the file is accessed sequentially, the OS prefetches the next block while the current one is being elaborated. To prove the benefits of I/O prefetching in FSC level-wise algorithms, we conducted synthetic tests whose results are reported in the plot on the right hand side of Figure 1. In these tests, we read a file of 256 MB in blocks of 4KB, and used an SCSI Linux workstation with 256 MB of RAM. We artificially varied the per-block computation time, and measured the total elapsed time. The difference between the elapsed time and the CPU time actually used to elaborate all the blocks corresponds to the combination of CPU idle time and time spent on I/O activity. Note that an approximated measure of the I/O cost for reading the file can be deduced for null per-block CPU time ($x = 0$): in this case the measured I/O bandwidth is about 10MB/sec. As we increase the per-block CPU time, the total execution time does not increase proportionally, but remains constant up to a certain limit. After this limit, the computational grain of the program is large enough to allow the OS to overlap computation and I/O. Since the cost of the candidate counting task rapidly increases as we decrease the support threshold, we argue that for small supports level-wise FSC algorithms can be considered compute and not I/O bound.

The plots reported in Figure 2 show per-iteration execution times of **DCP**, $Apriori_{DP}$ and DHP. The two plots refer to tests conducted on the same dataset, for support thresholds equal to 0.75 and 0.50. **DCP** always outperforms the other algorithms due to its more efficient counting technique. The performance improvement is particularly impressive for small values of k. For example, for $s = 0.75$, the second iteration of **DCP** takes about 21 sec. with respect to 853 and 1321 sec. for DHP and $Apriori_{DP}$. The superior performances of **DCP** are observable also in the plots reported in Figure 3, which show the total execution time obtained running *Apriori*, DHP, $Apriori_{DP}$, and **DCP** as a function of the support threshold. In these tests we varied some parameters of the synthetic datasets such as the average transaction size t, the total number m of items present in $\mathcal{D}$, and the number of transactions. In all the cases **DCP** remarkably outperformed its competitors. In many cases the performance improvement is of more than one order of magnitude. Moreover, for very small supports (0.25%), some tests with the other algorithms were not able to allocate all the memory needed. **DCP**, in fact, requires less memory than its competitors which exploit a hash tree for counting candidates, and is thus able to handle very low supports without the *explosion* of the size of the data structures used.

4 Related Work

The algorithms of the *Apriori* family [3,4,10,15] adopt a a level-wise behavior, which involves a number of dataset scans equal to the size of the largest frequent itemset. The frequent itemsets are identified by using a *counting-based* approach, according to which, for each level k, we count the candidate k-itemsets that are set-included in each transaction. An alternative approach, used by several other algorithms, is the *intersection-based* one [7,17,19]. In this case the database is stored in a vertical layout, where each record is associated with a distinct item and is stored as a list of transaction identifiers (*tid-list*). *Partition*, an *intersection-based* algorithm that solves several FSC *local* problems on distinct partitions of the dataset is discussed in [17]. Dataset partitions are chosen which are small enough to fit in the main memory, so that all these local FSC problem can be solved with a single dataset scan. While, during the first scan, the algorithm identifies a superset of all the frequent itemsets, a second scan is needed to compute the actual global support of all the itemsets. This algorithm, despite the small I/O costs, may generate a superset of all the frequent itemsets which is too large due to data skew, thus making the next iteration of the algorithm very expensive in terms of time and space. In [12] some methods to reduce the effects of this problem are discussed. Dataset sampling [18,20] as a method of reducing computation and I/O costs has also been proposed. Unfortunately, the FSC results obtained from a sampled dataset may not be accurate since data patterns are not precisely represented due to the sampling process. As discussed in Section 3, for small support thresholds, the *Apriori* family of algorithms becomes *compute-bound*, so that techniques as those illustrated in [5] can be effectively exploited to overlap I/O time. Since algorithms that reduce dataset scans [12,17,18] increase the amount of work carried out at each iteration, we argue that further work has to be done to quantitatively analyze the advantages and disadvantages of adopting these algorithms rather than level-wise ones.

Recently, several new algorithms for solving the FSC problem have been proposed [19]. Like *Partition*, they use an intersection-based approach by dynamically building a vertical layout database. While the *Partition* algorithm addresses these issues by relying on a blind subdivision of the dataset, Zaki's algorithms exploit clever dataset partitioning techniques that rely on a lattice-theoretic approach for decomposing the search space. However, Zaki obtains the best computational times with algorithms which mine only the maximal frequent itemsets (e.g. *MaxEclat, MaxClique*). While it is simple to derive all the frequent itemsets from the maximal ones, the same does not hold for their supports, which require a further counting step. Remember that the exact supports of all the frequent itemsets are needed to compute association rule confidences.

In [6] the Max-Miner algorithm is presented, which aims to find maximal frequent sets by looking ahead throughout the search space. This algorithm is

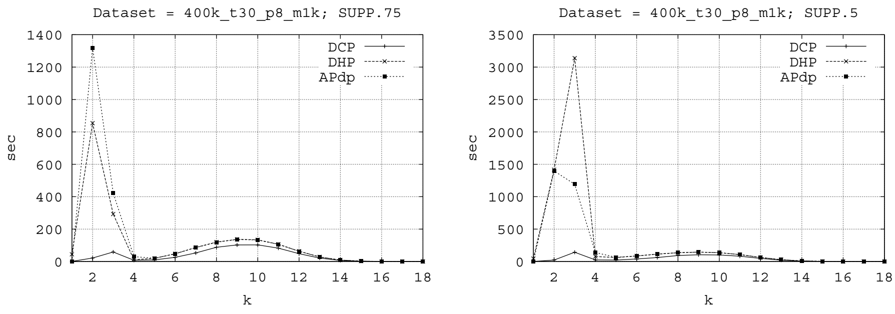


Fig. 2. Per-iteration execution times of DHP, $Apriori_{DP}$, and **DCP**.

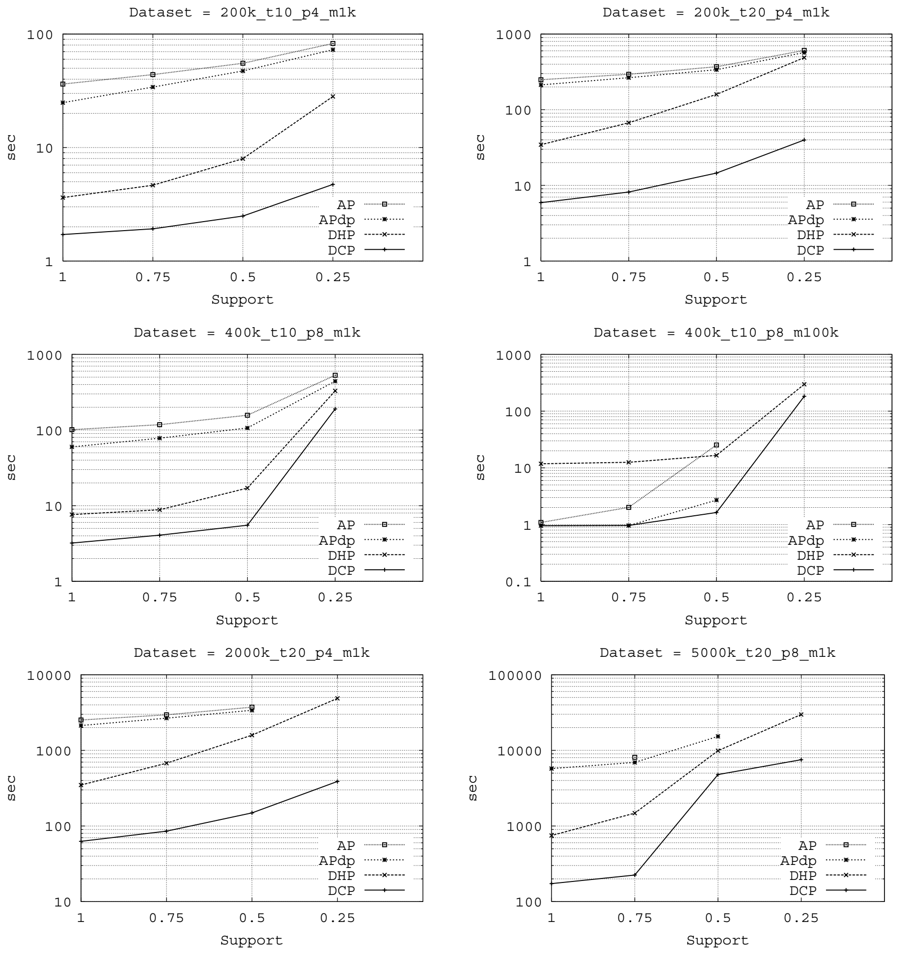


Fig. 3. Total execution times for *Apriori*, DHP, $Apriori_{DP}$, and **DCP** on different datasets as a function of the support threshold.

explicitly devised to work well for problems characterized by long patterns. When the algorithm is able to quickly find a long frequent itemset and its support, it prunes the search space and saves the work to count the support of all the subsets of this long frequent itemset. Moreover, it uses clever lower bound techniques to determine whether an itemset is frequent without accessing the database and actually counting its support. Hence, in this case too the method is not able to exactly determine the support of all frequent itemsets.

FP-growth [11], a very interesting algorithm able to find frequent itemsets in a database without candidate generation, has been recently presented. The idea is to build in memory a compact representation of the dataset where repeated patterns are represented once along with the associated repetition counters. The data structure used to store the dataset is called *frequent pattern tree*, or FP-tree for short. The tree is then explored without using the candidate generation method, which may explode for problems that have long itemsets. The algorithm is recursive. It starts by identifying paths on the original tree which share a common prefix. These paths are intersected by considering the associated counters. During its first steps, the algorithm determines long frequent patterns. Then it recursively identify the subsets of these long frequent patterns which share a common prefix. Moreover, the algorithm can also exactly compute the support of all the frequent patterns discovered. The authors only present results for datasets where the maximal frequent itemsets are long enough. We believe that for problems where the maximal frequent sets are not so long, the resulting FP-tree should not be so compact, and the cost of its construction would significantly affect the final execution time. A problem that has been recognized for FP-growth is the need to maintain the tree in memory. Solutions based on a partition of the database, in order to permit problem scalability, are also illustrated.

Finally we discuss Tree Projection [1], an algorithm which, like **DCP** prunes the transactions before counting the support of candidate itemsets, and also adopts a counting-based approach. The pruning technique exploited by **DCP** takes into account global properties regarding the frequent itemsets currently found, and produces a new pruned dataset at each iteration. Tree Projection, on the other hand, prunes (or projects) each transaction in a different way whenever the support of a specific group of candidates has to be counted. To this end candidates are subdivided into groups, where each group shares a common prefix. To determine the various projections of each transaction at level k, the transaction has to be compared with each frequent itemset at level $k-2$. These itemsets determine the common prefix of a given group of candidates of length k. This is the most expensive part of the algorithm. Once each projection has been determined, counting is very fast, since it only requires access to a small matrix of counters. The counting method used by **DCP** is different from Tree Projection. We need to access different groups of candidates (and associated counters) which share a common 2-item prefix. However, we do not need to scan all the possible 2-item prefixes. On the other hand, for each pair of items occurring in each transaction t we look up a directly accessible prefix table of C_k, and then scan the group of candidates to determine their inclusion in t. Note that the cost of this scan, due to the pruning operated on t, and the technique used to check the inclusion of each candidate in t, is very low.

5 Conclusions

In this paper we have reviewed the *Apriori* class of algorithms proposed for solving the FSC problem. These algorithms have often been criticized because of their level-wise behavior, which requires a number of scans of the database equal to the cardinality of the largest frequent itemset discovered. However we have shown that, in many practical cases, *Apriori*-like algorithms are not I/O-bound. When this occurs, the computational granularity is large enough to take advantage of the features of modern OSs, which allow computation and I/O to be overlapped. To speed up these compute-bound algorithms, it is thus important to make their most expensive computational part, i.e. the subset counting step, more efficient. Moreover, as the DHP algorithm demonstrates, counting the number of dataset scans as a measure of the complexity of the *Apriori* algorithms does not take into account that very effective dataset pruning techniques can be devised. These pruning techniques can rapidly reduce the size of the dataset until it fits into the main memory. Nevertheless, our results showed that the efforts to reduce the size of the dataset and the number of candidates are of little use if the subset counting procedure is not efficient.

Our ideas to design **DCP**, a new algorithm for solving the FSC problem, originate from all the above considerations. **DCP** uses effective database pruning techniques which, differently from DHP ones, introduce only a limited overhead, and exploits an innovative method for storing candidate itemsets and counting their support. Our counting technique exploits spatial locality in accessing the data structures that store candidates and associated counters. It also avoids complex and expensive pointer dereferencing. As a result of its design, **DCP** significantly outperforms both DHP and *Apriori*. For many datasets the performance improvement is even more than one order of magnitude. Due to its counting efficiency and low memory requirements, **DCP** exhibits good scalability and can thus be used to efficiently find low-support frequent itemsets within very large databases characterized by short or medium length patterns. Interested readers can find in [14] more details on **DCP**, as well as its pseudo code.

References

1. R. C. Agarwal, C. C. Aggarwal, and V.V.V. Prasad. Depth first generation of long patterns. In *Proc. of the 6th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages 108–118, Boston, MA, USA, 2000.
2. R. Agrawal, T. Imielinski, and Swami A. Mining Associations between Sets of Items in Massive Databases. In *Proc. of the ACM-SIGMOD 1993 Int'l Conf. on Management of Data*, pages 207–216, Washington D.C., USA, 1993.
3. R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. Inkeri Verkamo. Fast Discovery of Association Rules in Large Databases. In *Advances in Knowledge Discovery and Data Mining*, pages 307–328. AAAI Press, 1996.

4. R. Agrawal and R. Srikant. Fast Algorithms for Mining Association Rules in Large Databases. In *Proc. of the 20th VLDB Conf.*, pages 487–499, Santiago, Chile, 1994.
5. R. Baraglia, D. Laforenza, S. Orlando, P. Palmerini, and R. Perego. Implementation issues in the design of I/O intensive data mining applications on clusters of workstations. In *Proc. of the 3rd Workshop on High Performance Data Mining, in conjunction with IPDPS-2000, Cancun, Mexico*, pages 350–357. LNCS 1800 Spinger-Verlag, 2000.
6. R. J. Bayardo Jr. Efficiently Mining Long Patterns from Databases. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 85–93, Seattle, Washington, USA, 1998.
7. Brian Dunkel and Nandit Soparkar. Data organization and access for efficient data mining. In *Proceedings of the 15th ICDE Int. Conf. on Data Engineering*, pages 522–529, Sydney, Australia, 1999. IEEE Computer Society.
8. U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors. *Advances in Knowledge Discovery and Data Mining.* AAAI Press, 1998.
9. V. Ganti, J. Gehrke, and R. Ramakrishnan. Mining Very Large Databases. *IEEE Computer*, 32(8):38–45, 1999.
10. E. H. Han, G. Karypis, and Kumar V. Scalable Parallel Data Mining for Association Rules. *IEEE Transactions on Knowledge and Data Engineering*, 12(3):337–352, May/June 2000.
11. J. Han, J. Pei, and Y. Yin. Mining Frequent Patterns without Candidate Generation. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 1–12, Dallas, Texas, USA, 2000.
12. J.-L. Lin and M. H. Dunham. Mining association rules: Anti-skew algorithms. In *Proceedings of the 14-th Int. Conf. on Data Engineering*, pages 486–493, Orlando, Florida, USA, 1998. IEEE Computer Society.
13. A. Mueller. Fast Sequential and Parallel Algorithms for Association Rule Mining: A Comparison. Technical Report CS-TR-3515, Univ. of Maryland, College Park, 1995.
14. S. Orlando, P. Palmerini, and R. Perego. The DCP algorithm for Frequent Set Counting. Technical Report CS-2001-7, Dip. di Informatica, Università di Venezia, 2001. Available at `http://www.dsi.unive.it/~orlando/TR01-7.pdf`.
15. J. S. Park, M.-S. Chen, and P. S. Yu. An Effective Hash Based Algorithm for Mining Association Rules. In *Proc. of the 1995 ACM SIGMOD International Conference on Management of Data*, pages 175–186, San Jose, California, 1995.
16. N. Ramakrishnan and A. Y. Grama. Data Mining: From Serendipity to Science. *IEEE Computer*, 32(8):34–37, 1999.
17. A. Savasere, E. Omiecinski, and S. B. Navathe. An Efficient Algorithm for Mining Association Rules in Large Databases. In *Proceedings of the 21th VLDB Conference*, pages 432–444, Zurich, Switzerland, 1995.
18. H. Toivonen. Sampling Large Databases for Association Rules. In *Proceedings of the 22th VLDB Conference*, pages 134–145, Mumbai (Bombay), IndiaA, 1996.
19. M. J. Zaki. Scalable algorithms for association mining. *IEEE Transactions on Knowledge and Data Engineering*, 12:372–390, May/June 2000.
20. M. J. Zaki, S. Parthasarathy, W. Li, and M. Ogihara. Evaluation of Sampling for Data Mining of Association Rules. In *7th Int. Workshop on Research Issues in Data Engineering (RIDE)*, pages 42–50, Birmingham, UK, 1997.

Mining Cyclically Repeated Patterns

Ismail H. Toroslu[1] and Murat Kantarcioglu[2]

[1] University of Central Florida, Department of Computer Science,
Orlando, FL 32816-162362
toroslu@cs.ucf.edu

[2] Purdue University, Department of Computer Science,
West Lafayette, IN 47907-1398
kanmurat@cs.purdue.edu

Abstract. In sequential pattern mining, the support of the sequential pattern for the transaction database is defined only by the fraction of the customers supporting this sequence, which is known as the customer support. In this paper, a new parameter is introduced for each customer, called as repetition support, as an additional constraint to specify the minimum number of repetitions of the patterns by each customer. We call the patterns discovered using this technique as *cyclically repeated patterns*. The additional parameter makes the new mining technique more efficient and also helps discovering more useful patterns by reducing the number of patterns searched. Also, ordinary sequential pattern mining can be represented as a special case of the cyclically repeated pattern mining. In this paper, we introduce the concept of mining cyclically repeated patterns, we describe the related algorithms, and at the end of the paper we give some performance results.

1 Introduction

Several different forms of data mining problems have been investigated, and many different techniques have been developed. The original form of the sequential pattern mining problem started to get new attention [8,9,12], in addition to the early works on this problem [3,10]. The roots of the sequential pattern mining problem can be traced back to AI (as in [5]) and string processing (as in [11]). However, it got more attention from the database community after its redefinition in [3] as a database problem.

Sometimes, sequential pattern mining is treated under more general class of time-series analysis [6]. The time-series analysis problem had received more attention from the researchers. Some recent ones are presented in [1,4,7]. In this paper, we extend the very basic form of the sequential patterns to the cyclically repeated patterns, which is the generalization of the former one. The technique presented in [10] also generalizes the sequential pattern mining by considering temporal information on the transactions.

In order to define the concept of mining cyclically repeated patterns, a new parameter, called repetition support, is defined. In sequential pattern mining, there is only a single support constraint. Due to the additional constraint, in cyclically repeated pattern mining, the whole search process becomes much more efficient. In cyclically repeated pattern mining technique, since there will be less number of candidates compared to the sequential pattern mining, the memory requirement for the hash-table will significantly be reduced. Also, because of the additional constraint, there will be much less number of possible cyclically repeated patterns, and the algorithm will terminate more quickly. Data mining is basically used for decision support process. Therefore, introducing a new parameter will give the decision maker more flexibility in obtaining different results with two different types of supports. Also by producing less number of patterns it could be easier for the decision maker to interpret the results of the mining process.

We will also use the definitions similar to [2,3] in order to introduce the "cyclically repeated pattern mining" problem. Since the audience of this field is familiar with the customer transaction databases, throughout the paper we will also use customer transaction database as an example. Customer transaction database is a relation consisting of the following fields: customer id, transaction time, and items purchased. Itemset is any set of items, which can be purchased. Sequence is an ordered list of itemsets denoted as $[i_1, i_2, ..., i_n]$ where each i_p is an itemset. Cyclically repeated pattern is just a repeated sequence. Concatenation the sequence x k times can be represented as x^k. Thus, if x is a sequence $[i_1, i_2, ..., i_n]$, then x^k is $[i_1, i_2, ..., i_n, i_1, i_2, ..., i_n, ..., i_1, i_2, ..., i_n]$, where the subsequence $[i_1, i_2, ..., i_n]$ is repeated k times. When we have a sequence x^k, we call x (i.e., $[i_1, i_2, ..., i_n]$) as a cyclically repeated pattern with k repetitions, or a cyclically repeated pattern with repetition support k.

An example rule, which can be discovered from the customer transaction database, may be like: *It is likely that customers' purchase of "coke and chips" twice, is followed by a purchase of "milk and cookies*". If such a pattern is observed in sufficiently many number of customers, which is repeated in sufficiently many times by each customer (in which the patterns can start at any point of the pattern), then we can claim the existence of this pattern as cyclically repeated pattern.

A cyclically repeated pattern $[i_1, i_2, ..., i_n]$ represents all cyclically repeated patterns $[i_r, ..., i_n, i_1, i_2, ..., i_{r-1}]$, for all the values of r between 1 and n. We call all cyclically repeated patterns represented by $[i_1, i_2, ..., i_n]$ as the family of the cyclically repeated patterns. We choose the cyclically repeated pattern with the highest repetition support from its family to represent that family. Thus, each cyclically repeated pattern with length n represents a family of n patterns. If the representative of the family of the cyclically repeated patterns has k repetitions in a single customer's transactions, then all other patterns in the family have at least k-1 repetitions in that customer's transactions. Therefore, assuming all the cyclically repeated patterns in that family having repetition support k from that customer is a very reasonable estimation of the repetition support for those patterns. By using only the representative of the family we reduce the number of candidates in all the steps of the process.

The definition of the customer support is also adopted from [3]. In our technique, customer support of a pattern represents the fraction of customers with that pattern having the required repetition support. Therefore, the new cyclically repeated pattern mining technique can also be used to discover sequential patterns by just choosing the repetition support as one and discarding the "pattern families" concept and representing each candidate by themselves.

Even though there are also some new techniques proposed for sequential pattern mining (as in [8,12]), we have used the original algorithm of [3] since it can easily be adapted to the cyclically repeated pattern mining problem. Other algorithms might also be modified for this purpose. However, the main point of this paper is not just to show an efficiency of a specific algorithm, but the efficiency of the whole new technique. Since in our approach the number of patterns discovered can be reduced using the repetition support, it works more efficiently than the sequential pattern mining technique in all steps. Also reducing the memory requirement contributes the execution time as well, since the memory requirements of these algorithms could be very large and main memory might not be sufficient and page swaps could be needed.

The rest of the paper is organized as follows: In section 2, the main steps of the algorithm are described. The main difference of cyclically repeated pattern mining technique from the sequential pattern mining is in the fourth phase, which we call repeated sequence phase in our algorithm, and it corresponds to the sequence phase of the sequential pattern mining technique of [3]. In section 3, we describe how the addition of the repetition support changes that phase. The details of this phase are given for processing single customer's transactions. Section 3 also describes how the algorithm works for the whole transaction database. In section 4, some performance results are given, and finally section 5 presents the conclusions.

2 General Cyclically Repeated Pattern Algorithm

In this section we define the main steps of our algorithm, which is actually based on the "a priori all" technique of [2,3]. We have extended the sequential pattern mining algorithm of [3]. Also we have made some modifications on the data structure in order to determine the repetition supports. Except the fourth phase, other phases of our technique are almost the same as the one in [3]. Below they are briefly defined. The details of these phases can be found in [3].

Phase 1: Sort Phase: The database is sorted using customer id as the major key and transaction time as the minor key.

Phase 2: Litemset Phase: The set of large item sets, satisfying the given supports, are determined. They are called as litemsets. These litemsets are mapped to contiguous integers after they are determined. The difference between this phase in our algorithm and the one in [3] is that, the litemsets in our technique should also satisfy the repetition supports for each customer. Extending the transformation phase of [3] with this additional support constraint is very simple. In order to increment the customer support count, the number of repetitions of the litemsets for each customer is determined. If the number of repetitions is greater than or equal to the required repetition support then the customer support is increased. As in [3], where litemsets correspond to large-1-sequences, the litemsets in our technique also correspond to large-1-cyclically-repeated-patterns.

Phase 3: Transformation Phase: This phase is the same as the one in [3]. The original database is transformed into a database of litemset identifiers by replacing each transaction by the set of all the litemsets in that transaction. Also, the transactions that do not appear in litemsets are removed from the database.

Phase 4: Repeated Sequence Phase: We used "a priori all" algorithm with some modification in this phase. The data structure has also been modified slightly. The details of this phase are given in the following sections.

Phase 5: Maximal Phase: Maximal cyclically repeated patterns are determined from the set of all large cyclically repeated patterns. This phase is also very similar to the one in [3]. Since each large cyclically repeated pattern represents a family of the patterns just a simple subsequence check among large cyclically repeated patterns is not enough for the containment check. The subsequence check is extended in order to consider the subsequences that can be obtained by wrapping around the pattern.

3 Repeated Sequence Phase

The repeated sequence phase utilizes the "a priori" approach defined in [3]. This simply means that through the iterations, using the cyclically repeated patterns generated just in the previous iteration, larger cyclically repeated patterns are generated. At iteration *i*, possible candidates of that iteration, or *large-i-candidates* are generated using the *large-(i-1)-cyclically repeated-patterns* constructed in the previous iteration. These candidates are stored in a hash-tree structure in order to efficiently search each customer transaction sequence to determine their repetition supports and by combining those supports to determine their customer supports. Those candidates having at least the required minimum supports become *large-i-cyclically repeated-patterns*. In our technique there are some additions to the hash-tree data structure, and there are also some modifications in the candidate generation process and the hash-tree search process. We will use the following customer transactions sequence, which represents the list of the set of the litemsets obtained after the transformation phase:

[{1}, {2}, {2, 3, 4}, {3, 4}, {1}, {2}, {2, 3, 4}, {3, 4}, {1}]

Assuming the minimum required repetition support is 2, after the first two iterations the repetition supports of large-2-cyclically repeated-patterns satisfying the minimum support 2 will be obtained as follows:

Large-2-Cyclically repeated-Pattern	**Support**
[1, 2]	2
[1, 3]	2
[1, 4]	2
[2, 3]	2
[2, 4]	2
[3, 4]	2

At this iteration, the reverses of the some of the large-2-cyclically repeated-patterns should also be generated as large-2-cyclically repeated-patterns. However, we only keep one pattern from each family of patterns with the highest support representing that family. When the length of the patterns is 2, there will be only a pair of the form [a, b] and [b, a] in a family. The cyclically repeated patterns in the above table cover all the patterns with the required repetition support. In our example, all the reverse sequences either have the same or one less repetition supports than the ones in the above table.

When the number of distinct litemsets is *k,* the maximum number of possible candidates with length *n* is *k!/(k-n)!*. Only *k!/(n.(k-n)!)* of them are needed to represent all of them. Therefore, we also reduce the number of the candidates (and therefore the size of the hash-tables) by the ratio of *n* (the length of the candidates) by keeping only one representative from each family of the patterns. In sequential pattern mining, all the candidates had to be represented in the hash-table. In this example, we have 4 distinct litemsets, and eight patterns can represent all possible 24 candidates that can be generated with length 3. No patterns from the families of [1, 4, 2] and [1, 3, 2] have any support above the required repetition support 2. Therefore we will obtain only 6 large-3-cyclically repeated-patterns after this iteration. Some of the patterns indirectly represented do not necessarily have to have the required repetition support, and they might have one less repetition support than their representatives. For example the pattern [3, 4, 2] represented by [2, 3, 4] has only repetition support 1. After this step the following repetition supports will be obtained for the patterns satisfying the repetition support:

Large-3-Cyclically repeated-Pattern	**Support**
[1, 2, 3]	2
[1, 2, 4]	2
[1, 3, 4]	2
[1, 4, 3]	2
[2, 3, 4]	2
[2, 4, 3]	2

We will describe the candidate generation process after the third iteration using the same example. This process is different than the one in [3] since we use only a representative of a family to represent all of the possible cyclically repeated patterns in that family. In order to generate the candidates with "length n+1" for the next iteration, n many "length n-1" subsequences of each large-n-cyclically repeated-patterns are searched for joining. A candidate with "length n+1" is obtained by joining two large-n-cyclically repeated-patterns found in the previous iteration (the iteration n) by using their common "length n-1" subsequences, and then by adding the remaining two litemsets of these two patterns. For the above example, from "length 3" to go to the next iteration with "length 4" we check the followings subsequences:

Large-3-Cyclically-Repeated-Pattern	**Subsequences**
[1, 2, 3]	[1, 2], [2, 3], [3, 1]
[1, 2, 4]	[1, 2], [2, 4], [4, 1]
[1, 3, 4]	[1, 3], [3, 4], [4, 1]
[1, 4, 3]	[1, 4], [4, 3], [3, 1]
[2, 3, 4]	[2, 3], [3, 4], [4, 2]
[2, 4, 3]	[2, 4], [4, 3], [3, 2]

Since each pattern represents a family of patterns (a pattern with length n represents n patterns) we should consider the whole family in order to determine the candidates of the next iteration. For example the patterns [1, 2, 3] and [2, 3, 4] can be used to generate candidates [2, 3, 1, 4] and [2, 3, 4, 1]. Here the pattern [2, 3, 1] is considered from the family represented by [1, 2, 3]. Among the subsequences of [2, 3, 1, 4], [1, 4, 2] does not have the required repetition support, which has been determined at the previous iteration. Therefore, it will be discarded from the candidate list of the next iteration.

During the generation of the new "length n+1" candidates for the next iteration, each new candidate should be checked if a previously generated candidate already represents it, and if so, that candidate is discarded. Among all these large-4-cyclically repeated-candidates, assuming they are generated in the order shown in the above table, after removing the ones represented by previously generated candidates, we will only have the following 6 candidates at this iteration: [1, 2, 3, 4], [1, 2, 4, 3], [3, 1, 4, 2], [4, 1, 3, 2], [3, 4, 2, 1], and [4, 3, 2, 1].

After this step, we should also remove those large-4-candidates having at least one large-3-subsequence that does not have enough support. Each large-n-cyclically repeated-candidate has n *large-(n-1)-subsequence*; some of them are obtained by wrapping around the pattern. Again, for the subsequences we have to look for, if their representatives have required supports (in our example the required support is 2). Actually all of the "length n-1" subsequences (total n of them) can be obtained, each time by deleting a different litemset from "length n" candidate, n times. The following table shows for our example, the supports of the candidates from their subsequences:

Candidate	Subseqs with Support	Subseqs w/o Support
[1, 2, 3, 4]	[1, 2, 3], [2, 3, 4], [3, 4, 1], [4, 1, 2]	
[1, 2, 4, 3]	[1, 2, 4], [2, 4, 3], [4, 3, 1], [3, 1, 2]	
[3, 1, 4, 2]	[3, 1, 4], [4, 2, 3], [2, 3, 1]	[1, 4, 2]
[4, 1, 3, 2]	[4, 1, 3], [3, 2, 4], [2, 4, 1]	[1, 3, 2]
[3, 4, 2, 1]	[3, 4, 2], [1, 3, 4]	[4, 2, 1], [2, 1, 3]
[4, 3, 2, 1]	[4, 3, 2], [1, 4, 3]	[3, 2, 1], [2, 1, 4]

Since only candidates [1, 2, 3, 4] and [1, 2, 4, 3] have all their subsequences supported in the previous iteration, they will be the only two candidates that will be considered in this iteration. For our example customer sequence we will get repeated support 2 for both candidates making them as large-4-cyclically repeated-patterns.

Next, we will use the following sample database in order to explain the remaining details of the repeated sequence phase.

Customer 1: [{1}, {1}, {3}, {1}, {3}, {2}, {4}, {2, 3}, {1}, {3, 4}]
Customer 2: [{3, 4}, {1}, {1, 2, 3, 5}, {2, 4}, {1, 2, 5}]
Customer 3: [{2} {1, 4}, {3, 4}, {2, 3}, {1}, {4}, {2, 4}, {1, 2, 3, 5}, {1}, {1, 2, 3, 4, 5}]
Customer 4: [{4}, {2, 3}, {1, 4}, {1, 2, 3, 5}, {1, 2, 4, 5}, {2, 3, 4}, {1, 2, 4, 5}, {4}]

On this database the following support values are used:

- Repetition support = 2 (which means a pattern must be repeated at least twice by a customer in order to be considered as supported).
- Customer support = 3 (which means there must be at least 3 different customers for the pattern satisfying the given repetition support).

Basically, we will use the customer support, as in the ordinary sequential pattern mining for eliminating patterns without sufficient supports. However, in determining the customer support, we also take repetition support of the pattern into account for each customer. Simply when we process a customer transaction we increment the customer support count only for the patterns that also satisfy the repetition support constraint.

Each one of the integers in this database represents a litemset with sufficient customer (and therefore repetition) support. In the repeated sequence phase of our algorithm, similar to the ordinary sequential mining techniques, maximal large cyclically repeated patterns are determined. In order to do this, at each iteration step, larger item sets should be generated with the required supports. In this section we will also use the term *support* to represent customer supports satisfying the repetition support conditions. Initially, these litemsets have the following supports:

Litemset	**Support**
1	4
2	4
3	4
4	4
5	3

In the next iteration large-2-cyclically repeated-patterns satisfying the both support conditions are determined. Note that many patterns are eliminated since they do not have sufficient customer supports with the required repetition supports. For example the pattern [1, 2] does not have enough repetition support for customer 1 and customer 2 (which is just 1), although the same pattern have repetition support 3 for the customer 3 and customer 4. Although the pattern [1, 2] has a repetition support 2 for customer 4, we actually say it has a repetition support 3, by using the repetition support of the pattern [2, 1] which is the representative of the family pattern since it has the highest support for the family of [1, 2] and [2, 1].

When only the representatives of the pattern families are used, we will have the following large-2-cyclically repeated-patterns with required supports:

Large-2-cyclically repeated-pattern	**Support**
[3, 1]	4
[4, 1]	3
[4, 2]	3

After all the patterns with 2 litemsets having sufficient supports are determined, in the next step, only one cyclically repeated pattern with 3 litemsets will be determined. That will be [3,1,4] with support 3. These iterations continue until no cyclically repeated pattern is found for some length. In our example since there is no cyclically repeated pattern with length 4 having required supports, the algorithm terminates after the third iteration.

There is only a small difference in the new hash-tree-insert algorithm from the one in [3]. In order to determine the repetition supports of the candidates we have to keep additional information for each candidate, namely *the count of the candidate* and *the ending transaction number* of its last occurrence in a currently processed customer transaction.

The hash-tree search algorithm has more modifications than the one in [3], which is used to find only sequential patterns. In the new hash-tree-search algorithm whenever a candidate is detected in a customer sequence, its repetition support count has to be incremented and its end transaction number of its last occurrence up to that transaction in a customer sequence has to be recorded The modifications in the hash-tree search algorithm in order to count all the occurrences of the candidates are as follows: The starting transaction number of the partially generated candidate that is being searched is also passed as a parameter to the search algorithm. Whenever the searched pattern is found as a candidate in the hash-tree, its last occurrence ending transaction number is compared to the starting transaction number of the currently being searched occurrence to determine whether the currently being searched occurrence had started after the last occurrence had ended. Only if that is the case, the counter of the candidate is incremented, and the ending transaction number of the candidate is updated as the current transaction number.

4 Performance Results

We have used one of the datasets of [3] in our performance evaluations (the dataset C10-T2.5-S4-I1.25 of [3] with 25000 customers and with customer support as %1). In this dataset there are only a few and very short cyclically repeated patterns (with length 2 only), and there is no cyclically repeated pattern with repetition support 4 or 5. Since there were not many cyclically repeated patterns in this dataset we have created a new datasets from C10-T2.5-S4-I1.25 by using the first 5000 customers and repeating its transactions several times. We present the results of one of these files with 4 repetitions in Figure 1. We called the new dataset as C10-T2.5-S4-I1.5-4. The rest of the parameters in these datasets have not been changed. Since the other phases in our technique and the one in [3] are the same, in the Figures 1, we present the evaluation times of the repeated sequence phases only after normalizing the execution times according to repetition support 1 as 100. The results of other test cases were also very similar after the normalizations.

As it can be seen from this figure, there is a sharp decrease in the execution times of the repeated sequence phase of the algorithm when the repetition support increases. Even for the repetition support 2 there is a significant decrease in all the cases. Although we did not obtain the results for the ordinary sequential pattern search algorithm, it is clear that it will take more time than the cyclically repeated pattern algorithm even with repetition support 1. In cyclically repeated pattern search technique there are always fewer candidates than the sequential pattern search technique, because of representing pattern families with only one pattern.

These results can be interpreted in a broad way. They do not only show the different behaviors of the execution times of two algorithms, but in general the overall contribution of the “cyclically repeated pattern mining” approach over “sequential pattern mining”. Since the main reason for this improvement is due to the decrease in the number of candidates searched, similar kind of improvement should be expected from other kind of implementations of these techniques (i.e. by modifying other sequential pattern mining algorithms for the cyclically repeated pattern mining).

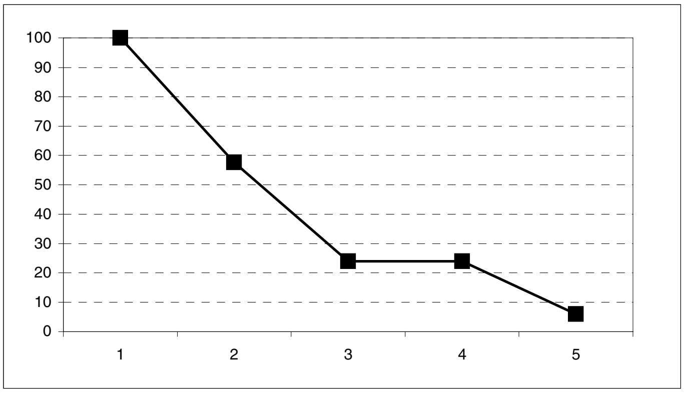


Fig. 1. Execution times of the repeated sequence phase, normalized according to repetition support 1, for the dataset C10-T2.5-S4-I1.5-4 with repetition supports 1 to 5 and customer supports 1%.

5 Conclusion

In this paper, we have presented an extension to the sequential data mining technique. We have defined the concept of cyclically repeated pattern mining by introducing a new support criterion. We have modified the data structures and the algorithms presented in [3] order to discover cyclically repeated patterns. We have also introduced the concept of the pattern families as a natural extension to the cyclically repeated pattern mining. The popular customer transaction database is used in order to describe the new technique.

Our performance results show that, searching for cyclically repeated patterns with higher repetition supports significantly decrease the execution time. In most cases, cyclically repeated pattern mining approach will discover less number of patterns than the sequential pattern mining, because, the repetition support introduces an extra constraint for the patterns, and also representing the family of patterns with a single pattern reduces the possible number of patterns. This affects positively all the steps of the mining process by decreasing the number of candidates and by reducing the sizes of hash trees. As a result, in cyclically repeated pattern mining technique the memory requirement is also reduced.

References

1. R. Agrawal, C. Faloutsos, A. Swami: "Efficient similarity search in sequence databases", *Proc. of the 4th Int'l Conference on Foundations of Data Organization and Algorithms,* Chicago, Oct. 1993, Also in Lecture Notes in Computer Science 730, Springer Verlag, pp. 69-84, 1993.

2. R. Agrawal, T. Imielinski, and A. Swami, "Mining association rules between sets of items in large databases", *Proc. of the ACM SIGMOD Conference on Management of Data,* pp. 207-216, Washington D.C., May 1993.
3. R. Agrawal, R. Srikant, "Mining sequential patterns", *Proc. of the Int'l Conference on Data Engineering (ICDE)*, pp. 3-14, Taipei, Taiwan, March 1995.
4. C. Bettini, X. S. Wang, S. Jajodia, "Mining temporal relationships with multiple granularities in time sequences", *Data Engineering Bulletin*, Vol. 21, pp. 32-38, 1998.
5. T. G. Dietterich, R.S. Michalski, "Discovering patterns in sequence of events", *Artificial Intelligence*, vol. 25, pp. 187-232, 1985.
6. J. Han, "Data mining", *Encyclopedia of Distributed Computing,* Eds. J. Urban and P. Dasgupta, Kluwer Academic Publishers, 1999.
7. J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, M-C. Hsu, Freespan: Frequent pattern-projected sequential pattern mining", *In Proc. 2000 Int. Conf. Knowledge Discovery and Data Mining (KDD'00)*, pp. 355-359, Boston, MA, Aug. 2000.
8. J. Han, J. Pei, Y. Yin, "Mining frequent patterns without candidate generation", *In Proc. 2000 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'00)*, pp. 1-12, Dallas, TX, May 2000.
9. M. Garofolakis, R. Rastogi, K. Shim, "Spirit: Sequential pattern mining with regular expression constraints", *In Proc. 1999 Int. Conf. Very Large Data Bases (VLDB'99)*, pp. 223-224, Edinburgh, UK, Sept. 1999.
10. R. Srikant, R. Agrawal, "Mining sequential patterns: generalizations and performance improvements". pp. 3-17, *Proc. of the Fifth Int'l Conference on Extending Database Technology (EDBT)*, Avignon, France, March 1996.
11. S. Wu, U. Manber, "Fast text searching allowing errors", Communications of the ACM, vol. 35, no. 10, pp. 83-91, Oct. 1992.
12. M. J. Zaki, "Efficient Enumeration of Frequent Sequences", *7th International Conference on Information and Knowledge Management*, pp 68-75, Washington DC, November 1998.

Development of Automated Data Mining System for Quality Control in Manufacturing

Hideyuki Maki and Yuko Teranishi

Systems Development Laboratory, Hitachi, Ltd.,
890 Kashimada, Saiwai-ku, Kawasaki-shi, Kanagawa-ken, Japan
{maki, teranisi}@sdl.hitachi.co.jp

Abstract. The production process in manufacturing has recently become highly complex. Therefore, it is difficult to solve problems in a process, by only using techniques that depend on the knowledge and know-how of engineers. Knowledge discovery in databases (KDD) techniques are supposed to assist engineers in extracting the non-trivial characteristics of a production process that are beyond their knowledge and know-how. However, the KDD process is basically a user-driven task and such a user-driven manner is not efficient enough for use in a manufacturing application. We developed an automated data-mining system designed for quality control in manufacturing. It has three features; periodical-analysis, storing the result, and extracting temporal-variances of the result. We applied it to liquid crystal display fabrication and found that the data-mining system is useful for the rapid recovery from problems of the production process.

1 Introduction

There is an emerging need for data-oriented knowledge discovery techniques in many fields of industry, where techniques that are based on human knowledge and know-how have so far been efficient enough to solve the problems encountered.

Manufacturing quality control involves monitoring the quality measures of products. When anomalous values are observed, process engineers analyze the data, identify the cause of the anomaly and modify the production process to eradicate it. In the conventional analytical process, engineers hypothesize the cause of an anomaly using the knowledge and know-how they have acquired, and examine the data using methods such as statistical analysis.

Recently, in manufacturing, the generations of products change rapidly and a variety of products are being developed in a short time. Also the production process is becoming much more highly complex, especially in the field of electronic fabrication, such as semiconductors. Thus, engineers hardly have enough time to acquire sufficient knowledge and know-how, thus making it difficult for them to solve problems.

Data-oriented knowledge discovery techniques, such as knowledge discovery in databases (KDD) are necessary in such situations. KDD is an iterative process of discovery and verification of the patterns in a large amount of data[1][2].

KDD techniques are supposed to assist engineers in extracting non-trivial characteristics from the data in a production process that is beyond their knowledge and know-how, and to contribute to advanced quality control in manufacturing. However, the KDD process is basically a user-driven task and such a user-driven manner is not efficient enough for use in a manufacturing application. Since the production process in a factory is usually automated and manufacturing data is obtained continuously from the process, the data must be monitored without interruption so that any anomaly of the process may be detected and eradicated immediately. Therefore, the KDD process must run concurrent with the production process. We have developed an automated data-mining system for the analysis of manufacturing data. This paper presents the characteristics of our data-mining system and reports the application of the system to the quality control of liquid crystal display (LCD) manufacturing.

2 Data Mining for Manufacturing Data Analysis

The KDD process generally consists mainly of two steps, which are "discovery" and "verification". In quality control, hypothesizing the cause of problems of a production process corresponds to the discovery step, and a detailed data analysis corresponds to the verification step. This detailed analysis of data is still the principal task of quality control, and the knowledge and know-how of the engineers are required for it. Thus, it is not our intention to automate the verification step. Our goal is to automate the discovery step of KDD. So, we developed an automated data-mining system — data mining is a typical class of analyzing methods used for the discovery task. The aim of our data-mining system is to clue engineers in on finding the causes of the problems in a production process. We discuss three characteristic functions of our data-mining system for the analysis of manufacturing data.

2.1 Periodical Data Feeding and Mining

The manufacturing data collected in a production process are stored in the universal databases at the factory which contain all information about the production process. In addition, the data-mining system has its own local database for storing the data to be mined, and the system transfers the data for mining from the universal databases to the local database. It is necessary in the data transfer task to determine which data should be transferred. One way to do it easily is to have the time of the last modification included in each record in the universal databases, and to transfer all the records which were modified after the last execution of the transferring task. The transferred data are combined with the data in the local database, and preprocessed in preparation for data mining.

The data transferred to the local database are the records of the events which occurred in the production process. A record consists of the time, the identification number (serial number) of the product, and other information

Serial#	Lot#	Process#	Machine#	EventCode	Date/Time	Faults
7W26AM	7W26	41	1821	start	11/22 19:42	
3W13AF	3W13	41	1822	start	11/22 19:56	
7W26AM	7W26	41	1821	finish	11/22 20:35	
7W26AR	7W26	90	2221	start	11/23 01:05	F002,F004
3W13AF	3W13	90	2221	start	11/23 01:35	F004

Fig. 1. Examples of event records

Serial#	Machine# for Process41	Machine# for Process42	Machine# for Process43	Fault F002	Fault F003	Fault F004	Time spent for Process41	Time spent for Process42	
7W26AM	1821	2400	0000	No	No	No	00:53	00:29	
3W13AF	1822	2401	2601	No	No	Yes	01:12	00:40	
7W26AR	1821	2402	0000	Yes	No	Yes	00:49	00:48	

Fig. 2. Examples of feature vectors

which is related to the event. Fig.1 shows some examples of records of events. The first example shows that the product having the serial number "7W26AM" was put into the manufacturing process identified as "process 41" using the machine whose number was "1821" at 19:42 on November 22. The last example shows that the product having the serial number "3W13AF" was put into process "90" using machine "2221" at 1:35 on November 23, which was an inspection process, and the product defect "F004" was detected. Note that there are multiple records of events corresponding to one product because multiple events occur to a product in the production process.

In the preprocess step for data mining, the data are transformed into the form where there is one record corresponding to one product. Each record consists of the attributes of each product, such as the identification numbers of machines used for manufacturing the product, the result of a fault inspection of it, and the time spent by in each process, as shown in Fig.2. Therefore, each record is regarded as a feature vector of a product. The feature vectors are stored in the local database of the data-mining system.

In addition to the transformation, a "categorization" process is performed in the preprocess step. In this process, numerical values in the feature vectors are categorized into a few classes and replaced with symbols representing the classes. For example, numerical values are categorized into three classes which are "large", "medium", and "small". Sometimes symbolic values in feature vectors are also categorized. The categorized feature vectors are also stored in the local database. Then, a set of the categorized feature vectors that have been prepared after the last execution of data mining is retrieved from the local database and fed into the data-mining engine. Subsequently, the main process of data mining, which is “rule extraction”, is put into execution.

We adopted the data-mining algorithm which we call “CHRIS” — characteristic rule induction by subspace search [3][4][5]. The data to be mined by the CHRIS algorithm is a data table that is retrieved from a relational database. We premise that each column of the table contains discrete values and that not so many, usually less than 50, cases of values are contained in a column. CHRIS requires a user to specify one target column, one target value in the target column, and multiple explanation columns. Then CHRIS search for the “if-then rules” which indicate the characteristics of the data. An if-then rule is described as “if $A1 = a1$ and $A2 = a2 \ldots$ then $B = b$”. $A1, A2, \ldots$ are some of the explanation columns and $a1, a2, \ldots$ are the values appearing in the columns. B is the target column and b is the target value. In principle, multiple predicates, i.e. sets of a column and a value, are allowed to appear in both the “if” part and the “then” part of a rule, however, we usually assume that a single predicate appears in the “then” part and limit the number of predicates in the “if” part to two or three.

The significance of each rule is evaluated with what we call a u-measure (utility measure). It is a product of two factors, which are the generality and accuracy of a rule. The measure for rule “if A then B” is defined as follows. (To simplify the notation here, we describe “$A1 = a1$ and $A2 = a2 \ldots$” as “A”, and “$B = b$” as “B”.)

$$u = P(A)^{\alpha} P(B|A) log \frac{P(B|A)}{P(B)} \tag{1}$$

where $P(A)$ is the probability that the predicate A is satisfied. $P(B)$ is that of predicate B, and $P(B|A)$ is the probability of the predicate B under the condition that the predicate A is satisfied. $P(A)^{\alpha}$ corresponds to the generality and $P(B|A)log[P(B|A)/P(B)]$ corresponds to the accuracy. Significant rules are expected to be both general and accurate. But, generally speaking, these two factors are mutually exclusive. Here, α is the trade-off parameter by which users can control the relative importance between them. We usually use $0 \le \alpha \le 1$. A larger value of α gives more importance to the generality. CHRIS searches all possible combinations of the predicates on the explanation columns for the rules whose u-measure values are larger than others. A list of the rules with large u-measure values is obtained as the result of a rule extraction.

This sequence of data feeding and mining is invoked automatically without a users' command in accord with a predetermined schedule.

2.2 Storage and Presentation of Data-Mining Results

Our data-mining system is able to archive data-mining results. The rule lists generated by data-mining method “CHRIS” are moved into the archives after the data-mining execution. The result of each data mining execution is saved in a file having its own unique name, including the date and the time of data-mining execution, for example, and these files are placed in the predetermined storage area.

Our data-mining system contains the directory file of the archives. When a data-mining result file is added in the archives, a new entry corresponding to the file is created and inserted into the directory file. Each entry in the directory file contains the name of the data-mining result file, and the date and time of the data-mining task. Users can access the archives and the directory file via the WWW system on the factory intranet. They refer to the directory file and find the data-mining result which they need to examine.

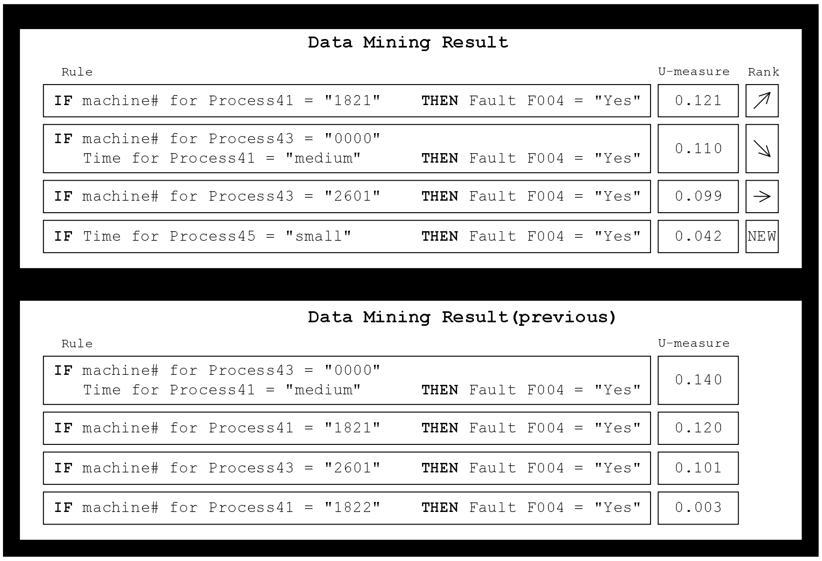
Because data mining is executed automatically, there may be time gaps between a data-mining execution and an examination of the results by a user. Thus, the next data-mining execution may start before the user examines the last result. Our system presents a data-mining result that a user intends to examine at any time they need. Also, the result of the last data-mining execution is provided on the WWW as the "last result". Just one click on a web browser brings the latest information to the user's screen.

2.3 Extraction of Temporal Variance

A production process must be stable in the commercial-production phase, hence, an irregular variance is undesirable. Even in a conventional production process, primary values measured in the process, such as the electric current and voltage in a circuit, are monitored and the engineers are notified of any irregular variances. However, the monitoring of only such primary values is not sufficient. The temporal change of implicit causality among the primary values is supposed to be informative for arriving at an understanding of what happens in a production process and why the primary values show an irregular variance. Since implicit causality is inferred from the results of data analysis, it is important to detect the changes of analysis results as well as the changes of values in the data. The difference between the results of two sequential data-mining executions is supposed to indicate the temporal variance of the characteristics of the data. Our system extracts the difference of two data-mining results and provides it for users as well as the result of each data-mining execution.

The rules extracted by data mining are listed in order of the u-measure values in a rule list. In temporal-variance extraction, the rank of each rule in the newer rule lists is compared with that of the corresponding rule in the older lists, and a change of rank is recognized as a "rise", a "fall", or a "stay". The symbol indicating the change of rank is added next to each rule in a rule list as shown in Fig.3. The rule list is stored in the archives together with the data-mining results. The users can refer to the time variance when they examine these results. The process of temporal-variance extraction is executed subsequent to the data mining.

The u-measure value of a rule represents its significance, therefore, those rules that are higher in rank are supposed to describe the more significant characteristics of the data. The rise in rank of a rule indicates that the characteristic represented by the rule is becoming more significant in the data. By using temporal-variance extraction, users are informed when there has been some vari-



Data Mining Result

Rule	U-measure	Rank
IF machine# for Process41 = "1821" **THEN** Fault F004 = "Yes"	0.121	↗
IF machine# for Process43 = "0000" Time for Process41 = "medium" **THEN** Fault F004 = "Yes"	0.110	↘
IF machine# for Process43 = "2601" **THEN** Fault F004 = "Yes"	0.099	→
IF Time for Process45 = "small" **THEN** Fault F004 = "Yes"	0.042	NEW

Data Mining Result(previous)

Rule	U-measure
IF machine# for Process43 = "0000" Time for Process41 = "medium" **THEN** Fault F004 = "Yes"	0.140
IF machine# for Process41 = "1821" **THEN** Fault F004 = "Yes"	0.120
IF machine# for Process43 = "2601" **THEN** Fault F004 = "Yes"	0.101
IF machine# for Process41 = "1822" **THEN** Fault F004 = "Yes"	0.003

Fig. 3. A rule list

ance of the characteristics in a production process to which they need to pay attention.

3 System Implementation

We have applied the data-mining system to LCD manufacturing. Fig.4 shows an overview of the system architecture. Our data-mining system consists of four modules, which are transfer, preprocess, rule extraction, and temporal-variance extraction. These modules are invoked sequentially on the predetermined schedule. The transfer module transfers the event records from the universal databases to the local database. The preprocess module transforms and categorizes the data in the local database to generate the data table to be mined, and stores it back in the local database. The rule-extraction module, i.e. the data-mining engine, retrieves the data table in the local database and extracts the rules representing the characteristics of the data. The rules are stored in the archives, and the directory file of the data-mining results is modified. The temporal-variance extraction module takes the rule lists generated in the last and the next to last rule extraction out of the archives, and generates a rule list that includes the temporal-variance information. The rule list is archived and the directory file of the data-mining results is modified, thus completing the data-mining process. The users can access the archives through WWW system on the factory intranet whenever they need to examine the data-mining results.

In the LCD factory, the data-mining process runs once a day at midnight and engineers can verify the latest data-mining results in the morning. The discovery

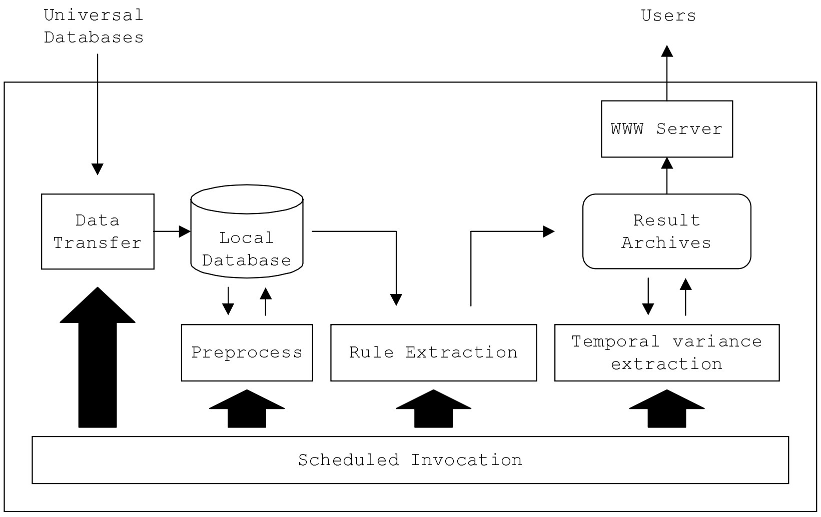


Fig. 4. An overview of the system architecture

step of KDD is performed by the data-mining system, and once engineers find the if-then rules that seem to indicate significant problems in the production process, they go next onto the verification step. They retrieve the related production process data in the universal databases and examine it in detail by means of data analysis tools to identify the cause of the problem.

4 Experimental Evaluation

We experimentally applied the data-mining system to an LCD fabrication production process. We used manufacturing data that was related to a fault in the process which had occurred a few months before. The cause of the fault was a problem in one of the machines used in a certain step of the production process. Let XX be the code of the machine with the problem, and AA be the code of the step where the machine XX was used. Actually, at that time, the fault rate of the LCD panels manufactured by machine XX was higher than that of the panels manufactured by the others in step AA. However, it had taken time for the engineers to find this fact out because there were a lot of steps they had to examine in the production process.

In the experiment, we used the data tables each of which contained the data obtained in a day indicating which machine had been used in each step of the production process and whether a fault had occurred for each LCD panel. The data-mining system searched for the rules representing the relationship between the occurrence of the fault and the machines used in the production process. Some of the rules obtained in data mining told us that “if a panel was processed using machine XX in step AA then the fault occurs", and they got higher in rank in the rule lists around the days when the fault of LCD panels had occurred.

The system intimated the cause of the product fault which the engineers had spent a lot of time searching for. Thus the data-mining system is considered useful for the rapid recovery from a fault of the production process.

5 Conclusion

We developed an automated data-mining system designed for data analysis in manufacturing. It has three typical features for analyzing manufacturing data. The first is the periodical execution of data feeding and mining, which enables automated data analysis concurrent with the production process. The second is storage and retrieval of data-mining results through WWW system on the factory intranet, by storing the archives of the data-mining results and providing them to users whenever they need them. The last is the temporal-variance extraction of data-mining result, which informs the users of the changes of implicit causality among the values observed in the manufacturing process. We experimentally applied the data-mining system to LCD fabrication, and tried searching for the cause of a product fault that had occurred a few months before. We analyzed the relationship between the occurrence of the fault and the machines used in the production process. The data-mining system found the relationship between the product fault and a specific machine, which the process engineers had spent much time searching for. Thus, our data-mining system is considered useful for the rapid recovery from a fault of the production process.

References

1. Fayyad, U., Piatetsky-Shapiro, G., Smyth, P.: Knowledge Discovery and Data Mining: Towards a Unifying Framework. Proc. KDD-96, AAAI Press (1996) 82–88
2. Piatetsky-Shapiro, G., Brachman, R., Khabaza, T.: An Overview of Issues in Developing Industrial Data Mining and Knowledge Discovery Applications. Proc. KDD-96 AAAI Press (1996) 89–95
3. Maeda, A., Maki, H., Akimori, H.: Characteristic Rule Induction Algorithm for Data Mining. Proc. PAKDD-98 (1998) 399–400
4. Ashida, H., Morita, T.: Architecture of Data Mining Server: DATAFRONT/Server. Proc. SCM-99 IEEE (1999) 882–887
5. Maki, H., Maeda, A., Akimori, H.: Data Mining Application to LSI Fault Analysis. Proc. ICEE-98, The Korean Institute of Electrical Engineers (1998) 417–420

FastFDs: A Heuristic-Driven, Depth-First Algorithm for Mining Functional Dependencies from Relation Instances

Extended Abstract

Catharine Wyss, Chris Giannella, and Edward Robertson

Computer Science Department, Indiana University, Bloomington, IN 47405, USA
{crood,cgiannel,edrbtsn}@cs.indiana.edu

Abstract. The problem of discovering functional dependencies (FDs) from an existing relation instance has received considerable attention in the database research community. To date, even the most efficient solutions have exponential complexity in the number of attributes of the instance. We develop an algorithm, FastFDs, for solving this problem based on a depth-first, heuristic-driven (DFHD) search for finding minimal covers of hypergraphs. The technique of reducing the FD discovery problem to the problem of finding minimal covers of hypergraphs was applied previously by Lopes *et al.* in the algorithm Dep-Miner. Dep-Miner employs a levelwise search for minimal covers, whereas FastFDs uses DFHD search. We report several tests on distinct benchmark relation instances involving Dep-Miner, FastFDs, and TANE. Our experimental results indicate that DFHD search is more efficient than Dep-Miner's levelwise search or TANE's partitioning approach for many of these benchmark instances.

1 Introduction

Functional dependencies (FDs) are a well-studied aspect of relational database theory. Originally, the study of FDs was motivated by the fact that they could be used to express constraints which hold on a relation schema *independently* of any particular instance of the schema (for example, a business rule). More recently, however, a new research direction for FDs has emerged: the *dependency discovery problem.* Given a relation schema, R, and an instance of the schema, r, we wish to determine all FDs which hold over r. This paper addresses this problem.

We develop an algorithm, *FastFDs*, for finding the canonical cover, $\mathcal{F}_r$, of the set of FDs of a given relation instance. FastFDs is based on a result in [9, 10], showing that finding $\mathcal{F}_r$ is equivalent to finding the minimal covers of each of a set of hypergraphs (one for each attribute) constructed from the *difference sets* of the relation instance.[1] We implemented FastFDs and tested it on several distinct classes of benchmark relation instances (§1.3).

[1] These were originally introduced in [9] and termed *necessary sets.*

1.1 Motivations

Two primary areas that motivate the dependency discovery problem are *data mining* and *data warehousing*. For example, paraphrasing from [6], consider a database of chemical compounds and their outcomes on a collection of bioassays. The discovery that an important property (such as carcinogenicity) of a compound depends functionally on some structural attributes can be extremely valuable. As another example, discovered FDs can provide valuable semantic information about data warehouses that can be used to save time in the evaluation of OLAP queries. For a more detailed description of motivations, see [13]. As an aside, we point out that Mannila *et al.* apply discovered FDs to database design ("Design-By-Example" [7]); however, the schema and instances involved remain small and thus most algorithms that solve the dependency discovery problem work quite well for this application.

1.2 Related Work

The first examination of the dependency discovery problem appeared in [9] (full version [10]), in which algorithms were presented for finding a cover for the set of discovered dependencies. The worst case output complexity of these algorithms is exponential in the size of the relation instance (although their behavior in practice was not investigated in [10]). This exponential output complexity was shown to be optimal in the sense that the number of dependencies in any cover can be exponentially larger than the size of the relation instance.

Since [9], research has focussed on developing algorithms that behave well in practice: [4,8,6], among others. Algorithms were developed, implemented and performance tested on both benchmark and synthetic data. Two recent algorithms are Tane [6] and *Dep-Miner* [8].[2]

Both Tane and Dep-Miner search the attribute lattice in a *levelwise* fashion. FastFDs differs from Tane in that: (1) Tane is not based on the hypergraph approach, and (2) Tane searches the attribute lattice in a levelwise fashion, whereas FastFDs uses a depth-first, heuristic-driven (DFHD) search strategy. FastFDs differs from Dep-Miner only in that Dep-Miner employs a levelwise search to find hypergraph covers, whereas FastFDs uses DFHD search.

1.3 Primary Contributions

The two primary contributions of this paper are as follows.

1. We introduce the algorithm FastFDs, which uses DFHD search to compute minimal FDs from a relation instance.
2. We report the results of several tests of the programs Dep-Miner, Tane, and FastFDs on three distinct classes of relation instances:
   a) Random integer-valued relation instances of varying correlation factors,
   b) random Bernoulli relation instances, and

[2] In [13] we report tests involving a third algorithm, *fdep* [4].

c) existing "real-life" relation instances available online at the Machine Learning (ML) Repository site [11].

Our experiments reveal that FastFDs is competitive for all of these relation instance classes when compared to TANE and Dep-Miner. Both FastFDs and Dep-Miner compute difference sets using the same technique. Omitting this computation, FastFDs is significantly faster than Dep-Miner on the three classes of relation instances (Tables 1(b), 2, Figure 2). Thus, DFHD search appears inherently more efficient than the levelwise calculation of covers of difference sets.

2 The Algorithm FastFDs

In this section, we describe the algorithm FastFDs. We begin with some fundamental definitions and results.

2.1 Definitions and Basic Results

Let R be a relation schema and r an instance of R.

Definition 1. *Let $X, Y \subseteq R$.*

1. *$X \to Y$ is a* functional dependency (FD) *over r iff for all tuples $t_1, t_2 \in r$, $t_1[X] = t_2[X] \Rightarrow t_1[Y] = t_2[Y]$. We write $r \vDash X \to Y$.*
2. *$X \to Y$ is* trivial *iff $Y \subseteq X$.*
3. *$X \to Y$ is* minimal *iff (i) $r \vDash X \to Y$ and (ii) $Z \subsetneq X$ implies $r \nvDash Z \to Y$.*

Definition 2. *The* canonical cover *for the set of FDs that hold over r is*

$$\mathcal{F}_r = \{X \to A \mid X \subseteq R, A \in R, r \vDash X \to A, A \notin X, \text{ and } X \to A \text{ is minimal}\}.$$

FastFDs assumes an alternate characterization of FDs, which relies on the concept of a *difference set.*

Definition 3. *1. For $t_1, t_2 \in r$, the* difference set *of t_1 and t_2 is $D(t_1, t_2) = \{B \in R \mid t_1[B] \neq t_2[B]\}$.*
2. *The* difference sets of r *are $\mathfrak{D}_r = \{D(t_1, t_2) \mid t_1, t_2 \in r,\ D(t_1, t_2) \neq \emptyset\}$.*
3. *Given $A \in R$, the* difference sets of r modulo A *are $\mathfrak{D}_r^A = \{D - \{A\} \mid D \in \mathfrak{D}_r$ and $A \in D\}$.*

The set $\mathfrak{D}_r^A$ provides an alternate characterization of FDs over r with RHS A.

Definition 4. *Let $\mathcal{P}(R)$ be the power set of R and $\mathcal{X} \subseteq \mathcal{P}(R)$. Then $X \subseteq R$* covers *$\mathcal{X}$ iff $\forall\ Y \in \mathcal{X},\ Y \cap X \neq \emptyset$. Furthermore, X is a* minimal cover *for $\mathcal{X}$ in case no $Z \subsetneq X$ covers $\mathcal{X}$.*

Consider $X \subseteq R$ that covers $\mathfrak{D}_r^A$. Let $D \in \mathfrak{D}_r^A$. Then $X \cap D \neq \emptyset$. Thus, X distinguishes any two tuples that disagree on A. On the other hand, this is exactly what it means that $r \vDash X \to A$. Thus, we have:

Lemma 1. *$r \models X \to A$ where $X \subseteq R$ iff X covers $\mathfrak{D}_r^A$.*

This lemma implies the main result of this section:

Theorem 1 ([9,10]). *Let $X \subseteq R$, $A \in R - X$, and r be a relation instance over R. $X \to A$ is a minimal functional dependency over r if and only if X is a minimal cover of $\mathfrak{D}_r^A$.*

Theorem 1 reduces the problem of computing $\mathcal{F}_r$ to the problem of finding minimal covers of $\mathfrak{D}_r^A$ for each attribute $A \in R$. In fact, any cover of $\underline{\mathfrak{D}_r^A} = \{D \in \mathfrak{D}_r^A \mid D' \in \mathfrak{D}_r^A \text{ and } D' \subseteq D \Rightarrow D' = D\}$ is also a cover of $\mathfrak{D}_r^A$. Thus, we need only retain *minimal* difference sets and compute minimal covers of $\underline{\mathfrak{D}_r^A}$. An efficient method for computing $\underline{\mathfrak{D}_r^A}$ is developed by Lopes *et al.* [8]. We employ a simpler version of this method [13].

The idea of computing $\mathcal{F}_r$ by computing minimal covers of $\mathfrak{D}_r^A$ was first identified in [9]. There, the authors note that the elements of $\mathfrak{D}_r^A$ form edges in a *hypergraph*; thus, the problem of computing $\mathcal{F}_r$ reduces to the problem of computing minimal covers for hypergraphs. A systematic study of complexity issues of many hypergraph problems relevant to computer science is given in [3]. A connection between a general framework for many data mining problems (including the dependency discovery problem) and the problem of finding hypergraph covers is studied in [5].

2.2 The FastFDs Algorithm

What follows is a brief sketch of the FastFDs algorithm. For detailed pseudo-code and a discussion of the complexity of the algorithm, see [13].

The main idea behind FastFDs is as follows. Every subset of $R - \{A\}$ is a potential candidate for a minimal cover of $\mathfrak{D}_r^A$. Consider a search tree which generates the subsets of $R - \{A\}$ in a depth-first, left-to-right fashion. Such a search tree is shown in figure 1(a). Each node in the tree represents a subset of $R - \{A\}$, given by the labeling along the path from the root to that node. There are $2^{|R|-1}$ such nodes. Note that in generating the subsets without repeats, we have "ordered" the attributes lexically: $B > C > D > E > F$.

Consider an instance r over R which gives rise to the set $\underline{\mathfrak{D}_r^A} = \{BC, BD, CF, DE\}$. At each node in our new search tree, we *order the remaining attributes according to how many remaining difference sets they cover.* We break ties in this ordering lexically. This search pattern is illustrated in Figure 1(b).[3] At each node in the search tree, we track the remaining difference sets and the ordering of the remaining attributes.

The method shown in Figure 1(b) is the search strategy used by FastFDs. Each leaf node indicates one of three base cases in our search.

1. If we arrive at a node where there are still difference sets left to cover, but no attributes are left to unfold, fail (there are no FDs down this branch) – see leaves 5, 7, and 8 in Figure 1(b)).

[3] Attributes which do not appear in any remaining difference set are not included at the next level.

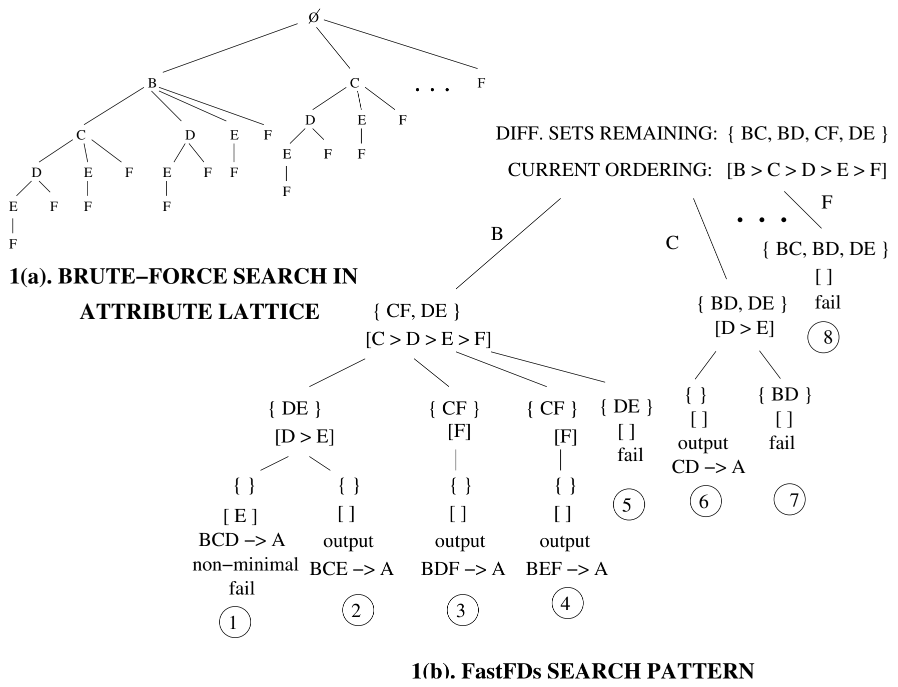


Fig. 1. FastFDs Example

2. If we arrive at a node where there are no difference sets left, this implies one of two situations.
   a) The FD may not be minimal; in this case, fail (leaf 1).
   b) Otherwise, output the subset built along the path to the current leaf as a left hand side (LHS) for an FD for the current attribute (leaves 2, 3, 4, and 6).

We use a simple, greedy heuristic at each node: search the attributes that cover the most remaining difference sets first. This heuristic works well in practice (see §3), but can be fooled into performing excess work. For example, in Figure 1(b) (leaf 1), the FD $BCD \rightarrow A$ is not minimal, since $CD \rightarrow A$ is also an FD (leaf 6). In this case, it is the way we break ties (lexically) that causes the extra branch; in general, the excess work is not straightforward to characterize.

One of the main advantages of FastFDs over its levelwise-search counterparts, Dep-Miner and TANE, is that the space complexity of FastFDs is limited to $O(|\mathfrak{D}_r| \cdot |R|)$ space. The levelwise approaches of TANE and Dep-Miner consume an amount of space directly proportional to the largest set of candidates generated in the levelwise search, s_{max}. Thus, FastFDs is significantly faster than the levelwise approach for large $|R|$, since the size of s_{max} can be exponential in $|R|$. This trend is especially manifested in the case of Bernoulli relation instances where the average length of FDs is predicted to be $|R|/2$ or larger.

3 Experimental Results

FastFDs has been implemented in C++. The program has been tested on a 700MHz Athlon system with 128MB RAM, running Red Hat Linux 6.1.[4] We have also implemented our own version of Dep-Miner, which uses the same code as FastFDs for reading input relations and calculating $\mathfrak{D}_r^A$. Our versions of both Dep-Miner and FastFDs run entirely in main memory. The version of TANE we use is available on the web [12].[5]

3.1 Integer-Valued Relation Instances

Our first set of experiments involved randomly-generated integer relation instances. Table 1 (below) illustrates the performance of TANE, Dep-Miner and FastFDs on such instances. Part (a) represents experiments involving all three algorithms with $|R|$ fixed at 20. Part (b) represents experiments between Dep-Miner and FastFDs in which the computation for generating the difference sets from r (genDiffSets) is not counted; $|r|$ is fixed at 10,000 in table 1(b). Each relation instance is generated according to a *correlation factor* (CF), as in [8]. The higher the CF, the fewer distinct values appear in each instance column.[6]

Table 1. Results on Random Integer-Valued Relation Instances

Table 1(a)

Instance	Time (seconds)					
	CF = 0.0		CF = 0.5		CF = 0.9	
$\vert r\vert$	Dep/Fast†	TANE‡	Dep/Fast†	TANE‡	Dep/Fast†	TANE‡
50,000	5	13	6	10	6	7
100,000	12	38	13	27	13	17
150,000	19	78	21	50	19	29
200,000	25	133	29	78	26	41
250,000	33	207	37	115	33	57
300,000	40	478	46	360	41	158
350,000	48	◦	69	476	54	196

†Dep-Miner and FastFDs took the same amount of time ($\pm 0.01s$).
‡TANE/MEM took approximately the same amount of time as TANE ($\pm 1s$).
◦ indicates TANE took longer than 2 hours and was aborted.

Table 1(b)

Instance	Time (seconds) with genDiffSets ommitted and $\vert r\vert = 10,000$					
	CF=0.0		CF = 0.5		CF = 0.9	
$\vert R\vert$	FastFDs	Dep-Miner	FastFDs	Dep-Miner	FastFDs	Dep-Miner
10	0*	0*	0*	0*	0*	0*
20	0*	0.1	0.1	0.2	0.1	0.2
30	0.2	0.3	0.8	1.6	1.0	2.1
40	0.6	1.2	3.7	9.5	5.5	11.0
50	2.3	3.8	14.3	27.5	20.8	48.0
60	5.6	8.6	41.6	81.2	64.7	134.4

*Less than $0.1s$.

[4] The source code for FastFDs is available from the authors upon request.
[5] We also indicate the performance of TANE/MEM in the following tests.
[6] The range of distinct values for each attribute is $(1 - CF) * num\_tuples$.

The results in Table 1(a) are in line with those reported in [8]. One interesting fact revealed by our tests is that less than 0.1% of the computation time for Dep-Miner and FastFDs is spent searching for FDs and over 99.9% is spent generating difference sets (genDiffSets). We see from Table 1(a) that Dep-Miner and FastFDs are 2-3 times faster than Tane for CFs from 0.0-0.9.

Table 1(b) illustrates the relative performance of FastFDs and Dep-Miner as $|R|$ varies from 10 to 60. Note that as the CF increases, FastFDs becomes progressively more efficient than Dep-Miner. As we will see in the next sections (§3.2, §3.3), random integer-valued relation instances with low CFs do not appear to be good predictors of performance on real-life instances such as those from the ML repository. Bernoulli instances appear notably better in this regard.

3.2 Bernoulli Relation Instances

Our second set of experiments involved randomly generated Bernoulli (i.e. two-valued) relation instances. Such relation instances are an interesting testbed for programs finding minimal FDs. Whereas random integer-valued instances seem amenable to the search methods of FastFDs, Dep-Miner and Tane, there tend to be few minimal FDs. This is not the case in Bernoulli instances. The average length of minimal FDs in random Bernoulli instances has been investigated and is straightforward to quantify [2]. The expected average length of minimal FDs is $|R|/2$ when $|r| = 2^{|R|/4}$. This length represents the maximal possible number of subsets at any one level in the power set lattice of R. In practice, the number of minimal FDs returned when $|r| = 2^{|R|/4}$ increases exponentially with $|R|$.

Since the case of minimal FDs of size $|R|/2$ involves relatively small-sized relation instances for $|R| < 30$, the impact of the $O(|R||r|^2)$ difference set calculation is minimized for Dep-Miner and FastFDs. Whereas in the random integer-valued instances in §3.1 less than 0.1% of the computation time of Dep-Miner and FastFDs is spent searching for FDs, in random Bernoulli instances where $|r| = 2^{|R|/4}$, over 99% of the time is spent searching for FDs (as opposed to computing difference sets).

Another reason this particular case is interesting is that, since there are so many subsets of size $|R|/2$, levelwise algorithms require exponential space to store the candidate sets at intermediate levels near to $|R|/2$. Thus, this test case allows us to see the impact of the exponential space usage of the levelwise algorithms, versus the $O(|R||r|^2)$ space usage of FastFDs.

Table 2 (left) illustrates the relative performance of Tane, Dep-Miner and FastFDs for the case of Bernoulli relation instances, where $|r| = 2^{|R|/4}$ and $|R|$ ranges from 20 to 28.

Relation Instance			Time (seconds)		
$\lvert R\rvert$	$\lvert r\rvert = 2^{\lvert R\rvert/4}$	$\lvert\mathcal{F}\rvert$	FastFDs	Tane[†]	Dep-Miner
20	32	7.6×10^4	2	5	195
21	38	1.4×10^5	5	15	601
22	45	2.6×10^5	11	36	1893
23	54	5.4×10^5	28	84	5169
24	64	1.0×10^6	69	392	14547
25	76	2.0×10^6	171	●	38492
26	91	3.9×10^6	428	●	●
27	108	7.4×10^6	1081	●	●
28	128	1.6×10^7	3167	●	●

[†] Tane/mem exhibited similar results.
● indicates program ran out of memory.

Fig. 2. Results on Bernoulli instances; $|r| = 2^{|R|/4}$

The number of FDs is increasing exponentially with $|R|$; thus it is not surprising that execution time does as well. However, here we can see the impact of the exponential space usage on TANE and Dep-Miner. For $|R| \geq 25$, execution of TANE was aborted after it ran out of memory.

The results for Dep-Miner are similar. Although Dep-Miner's space usage seems slightly better than TANE's, Dep-Miner's performance is over two orders of magnitude worse than FastFDs. In contrast, FastFDs was never in danger of running out of memory.[7] This is crucial, both because memory is a scarcer resource than CPU cycles, and because the CPU cycles are utilized much more efficiently when swapping is avoided.

3.3 Machine Learning Repository Relation Instances

Our third set of experiments involved non-synthetic relation instances extracted from the Machine Learning (ML) repository, online [11]. Table 2 (below) illustrates the performance of TANE, Dep-Miner and FastFDs on 9 ML repository instances.[8]

Table 2. Results on ML Repository Instances

ML Instance				Time (seconds)			
Name	$\|R\|$	$\|r\|$	$\|\mathcal{F}\|$	FastFDs	TANE	TANE/MEM	Dep-Miner
Chess	**7**	28,056	1	160	1	$0^\dagger$	160
Flare	**13**	1389	0	5	11	1	5
Adult	**15**	48,842	66	7269	1722	•	7269
Credit	**16**	690	1099	2	1	$0^\dagger$	6
Letter Recognition	**17**	20,000	61	1577	2487	•	1645
Lymphography	**19**	148	2730	4	41	9	91
Hepatitis	**20**	155	8250	3	13	5	212
Automobile	**26**	205	4176	1	179	81	○
Horse Colic	**28**	300	128,726	123	•	•	•

† indicates program took less than 0.01 seconds to complete.
○ indicates execution time exceeded 3 hours and was aborted.
• indicates program ran out of memory.

First, compare the running times of FastFDs and Dep-Miner. FastFDs meets or exceeds Dep-Miner's performance in all cases. As $|R|$ becomes larger, FastFDs becomes increasingly faster than Dep-Miner.

Now compare the running times of FastFDs and TANE. Here, the comparison is not as straightforward. For large $|r|$, note the lengthy times of FastFDs. This is due to the $|r|^2$ difference set calculation. On the other hand, as $|R|$ grows, TANE experiences serious difficulties due to excessive memory consumption. For

[7] Memory usage remained below 0.5MB for all $|R|$ shown.

[8] These tests were run on preprocessed, purely integer-valued versions of the ML relations, as in TANE. In addition, the three algorithms were tested on the following instances and each returned in less than 0.1 second: Liver, Abalone, Pima, Tic-Tac-Toe, Wisconsin Breast Cancer, Echocardiagram and Housing.

the Horse-Colic database ($|R| = 28$), FastFDs finds the minimal FDs quickly; Tane runs out of memory. The best method for finding minimal FDs in real-life situations might involve a tradeoff between Tane (for small $|R|$ and large $|r|$) and FastFDs (small $|r|$ or larger $|R|$).

4 Conclusions and Further Directions

We have presented a novel search method, FastFDs, for computing minimal FDs from difference sets using DFHD search. Our experimental results (§3) indicate that FastFDs is competitive for each of the following classes of benchmark relation instances: (1) random integer-valued instances of varying correlation factors, (2) random Bernoulli instances, and (3) real-life ML Repository instances. In fact, our experiments indicate that for $|R| \geq 17$, FastFDs is significantly faster for all classes, due to the inherent space efficiency of DFHD search. For $|R| < 17$, a better choice may be the Tane algorithm for class (3) instances.

It is interesting that the heuristic-based, depth-first search methods common in solutions to Artificial Intelligence (AI) problems are usually eschewed by the data mining community, because they have a tendency *not* to scale up to the massive amounts of data the data mining programs must be run on. However, our experiments show that for the case of computing minimal covers of $\mathfrak{D}_r^A$, in fact the DFHD search strategy fares significantly better than the canonical levelwise approach. In fact, for the case of Bernoulli databases, when the FDs computed are of average length $\geq |R|/2$, the heuristic-driven approach is astoundingly more efficient (§3.2).

The space efficiency of FastFDs makes it a natural algorithm for parallelization. One obvious multi-threaded version of FastFDs would involve delegating the search for FDs with RHS A to distinct processors for distinct attributes A.

The performance of FastFDs depends crucially on the simple, greedy heuristic we have chosen for computing minimal hypergraph covers. Our heuristic works for finding minimal covers for *general* hypergraphs; however, not every hypergraph can be realized from a relation instance. Additional constraints are implied by the relation instance which are not reflected in the current heuristic. Further study of these constraints with the aim of designing better heuristics seems an interesting path to pursue.

Another useful direction for further study (pointed out in [9]) is to consider the incremental dependency inference problem: given, r, the canonical cover of the set of dependencies $\mathcal{F}_r$, and a tuple t, find the canonical cover of $r \cup \{t\}$ $(r - \{t\})$. To the best of our knowledge, this problem has not yet been addressed, and seems reasonably challenging.

Finally, it seems reasonable that most applications of the dependency inference problem do not require all dependencies but only the "interesting" ones. For example, dependencies with small left-hand sides are likely to be more useful that ones with large left-hand sides. Characterizing the interestingness of dependencies and tailoring algorithms to finding these dependencies is a good direction for future work.

Acknowledgments. The authors thank the following individuals for contributions to this paper: Felix Wyss, Dennis Groth, Mehmet Dalkilic, Juha Kärkkäinen, Stéphane Lopes, and the anonymous DaWaK reviewers. The databases in §3.3 were obtained from the ML online Repository online [11]. The lymphography database was obtained from the University Medical Centre, Institute of Oncology, Ljubljana, Yugoslavia. Thanks go to M. Zwitter and M. Soklic for providing the data.

References

1. Agrawal, Rakesh; Mannila, Heikki; Srikant, Ramakrishnan; Toivonen, Hannu and Verkamo, A.I. "Fast Discovery of Association Rules." *Advances in KDD*, AAAI Press, Menlo Park, CA, pg. 307-328, 1996.
2. Demetrovics, J; Katona, G; Miklos, D; Seleznjev, O. and Thalheim, B. "The Average Length of Keys and Functional Dependencies in (Random) Databases." *Lecture Notes in Computer Science*, vol. 893, 1995.
3. Eiter, Thomas and Gottlob, Goerg. "Identifying the Minimal Traversals of a Hypergraph and Related Problems." *SIAM Journal of Computing*, vol. 24, no. 6, pg. 1278-1304, 1995.
4. Flach, Peter and Savnik, Iztok. "Database Dependency Discovery: a Machine Learning Approach." *AI Comm.* vol. 12, no. 3, pg 139-160.
5. Gunopulos, Dimitrios; Khardon, Roni; Mannila, Heikki; and Toivonen, Hannu. "Data Mining, Hypergraph Traversals, and Machine Learning (extended abstract)", PODS, 1997, pg 209-216.
6. Huhtala, Ykä; Kärkkäinen, Juha; Porkka, Pasi and Toivonen, Hannu. "TANE: An Efficient Algorithm for Discovering Functional and Approximate Dependencies." *The Computer Journal*, vol. 42, no. 2, 1999.
7. Kantola, Martti; Mannila, Heikki; Räihä, Kari-Jouko and Siirtola, Harri. "Discovering Functional and Inclusion Dependencies in Relational Databases." *Journal of Intelligent Systems*, vol. 7, pg. 591-607, 1992.
8. Lopes, Stephane; Petit, Jean-Marc and Lakhal, Lotfi. "Efficient Discovery of Functional Dependencies and Armstrong Relations." *Proceedings of ECDT 2000.* Lecture Notes in Computer Science, vol 1777.
9. Mannila, Heikki and Räihä, Kari-Jouko. "Dependency Inference (Extended Abstract)", *Proceedings of the Very Large Databases Conference (VLDB)*, Brighton, pg. 155-158, 1987.
10. Mannila, Heikki and Räihä, Kari-Jouko. "Algorithms for Inferring Functional Dependencies from Relations", *Data & Knowledge Engineering*, 12, pg. 83-99, 1994.
11. Merz, C.J. and Murphy, P.M. UCI Machine Learning databases (1996). `http://www.ics.uci.edu/~mlearn/MLRepository.html`. Irvine, CA: University of California, Department of Information and Comp. Sci.
12. The Tane and Tane/mem source code is available on the web at `http://www.cs.helsinki.fi/research/fdk/datamining/tane`.
13. Wyss, C; Giannella, C; and Robertson E. "FastFDs: A Heuristic-Driven, Depth-First Algorithm for Mining Functional Dependencies from Relation Instances", *Technical Report*, Dept. of Comp. Sci, Indiana University, May 2001.

Ensemble Feature Selection Based on the Contextual Merit

Seppo Puuronen, Iryna Skrypnyk, and Alexey Tsymbal

Department of Computer Science and Information Systems, University of Jyväskylä,
P.O. Box 35, FIN-40351 Jyväskylä, Finland
{sepi,iryna,alexey}@cs.jyu.fi

Abstract. Recent research has proved the benefits of using ensembles of classifiers for classification problems. Ensembles constructed by machine learning methods manipulating the training set are used to create diverse sets of accurate classifiers. Different feature selection techniques based on applying different heuristics for generating base classifiers can be adjusted to specific domain characteristics. In this paper we consider and experiment with the contextual feature merit measure as a feature selection heuristic. We use the diversity of an ensemble as evaluation function in our new algorithm with a refinement cycle. We have evaluated our algorithm on seven data sets from UCI. The experimental results show that for all these data sets ensemble feature selection based on the contextual merit and suitable starting amount of features produces an ensemble which with weighted voting never produces smaller accuracy than C4.5 alone with all the features.

1 Introduction

Many methods for constructing ensembles have been developed. Some of them are general methods while the others are specific to particular learning algorithms. Amongst successful general techniques are sampling methods that manipulate the training set, and methods that manipulate the output targets. The common approach is to use some heuristics to manipulate the training sets before construction of the base classifiers of an ensemble. Researchers in this area have developed and evaluated different criteria for ensemble goodness [5, 14, 19]. Generally, those criteria are defined in terms of diversity and accuracy. Many methods attempt to satisfy the criteria with a thoroughly selected heuristic for the construction of base classifiers. However, the goodness of an ensemble depends also on the integration method applied with the ensemble and thus feedback about the goodness of an ensemble may be taken benefit of during the base classifier construction process.

In this paper we propose one wrapper approach -based ensemble creation method that refines the goodness of an ensemble iteratively. A feature selection heuristic is used to manipulate the subsets of features of the training set used to construct individual base classifiers. The refinement cycle helps to fit the feature selection heuristic to the particular learning algorithm and data set used. We use the contextual feature merit (CM) measure [1] and the diversity of an ensemble as the evaluation function in our new algorithm. The integration method applied is weighted voting.

In Section 2 the base framework of an ensemble classification is considered. Section 3 describes the contextual merit measure as a criterion for ensemble feature selection and the algorithm realizing the wrapper approach for ensemble feature selection. In Section 4 our experimental study on several data sets is described. We summarize with conclusions in Section 5.

2 Ensemble Classification

In supervised learning, a learning algorithm is given training instances of the form $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_M, y_M)\}$ for some unknown function $y = f(\mathbf{x})$, where $\mathbf{x}_i$ values are vectors of the form $\langle x_{i,1}, \ldots, x_{i,j}\rangle$, where $x_{i,j}$ are feature values of $\mathbf{x}_i$, and M is the size of the training set $\mathbf{T}$. In this paper we will refer to features as f_j, $j = 1 \ldots N$, where N is the number of features. Given a set of training instances $\mathbf{T}$, a learning algorithm produces a *classifier*. The classifier is a hypothesis about the true function f. Given new $\mathbf{x}$ values, it predicts the corresponding y values. We will denote base classifiers by $h_1, \ldots, h_S$, where S is the size of ensemble. An *ensemble* of classifiers is a set of classifiers whose individual decisions are combined to classify new instances. Research has shown that an effective ensemble should consist of a set of base classifiers that not only have high accuracy, but also make their errors on different parts of the input space [5].

Classification process includes two phases: (1) *learning phase*, and (2) *application phase*. During the learning phase, the set of base classifiers is generated in some way. Each base classifier in the ensemble (classifiers $h_1 \ldots h_S$) is trained using training instances of the subsets $\mathbf{T}_i = (1 \ldots S)$ obtained after partitioning the initial training set $\mathbf{T}$. The results of the base classifiers are combined in some way $h^* = F(h_1, h_2, \ldots, h_S)$ to produce the final classification. At the application phase, a new instance $(\mathbf{x}, y)$ is given with the unknown value y to be classified by the ensemble. As a result, the class value y^* is then predicted as $y^* = h^*(\mathbf{x})$.

Combining classifications of several classifiers can be useful only if there is disagreement among the base classifiers, i.e. they are independent in the production of their errors [5]. The error rate of each base classifier should not exceed a certain limit. Otherwise, the ensemble error rate will usually increase as a result of combination of their classifications, as in the case of weighted voting [5]. One commonly used measure of independence of the base classifiers is the *diversity* of an ensemble [17].

In the framework of ensemble classification, there are two major issues: (1) the learning algorithm by which the base classifiers are generated, and (2) the algorithm by which the classifications are combined. A comprehensive review of the generation and combination algorithms is given in [5]. Algorithms for generating ensembles include sampling techniques, techniques that manipulate the input features, and techniques that manipulate the output targets, amongst others. In this paper we concentrate on manipulating the subsets of features of the training set. We will use the term *ensemble feature selection* for the task of varying the feature subsets used by each base classifier of an ensemble to promote diversity. The ensemble feature selection concept was first introduced in [13].

One commonly used algorithm for combining classifications is *weighted voting*. This algorithm is based on the voting principle, when each classifier gives some "vote" for assigning a particular class value to the input instance. The class value with the most votes is selected. In the case of weighted voting, the votes are weighted, usually by estimated accuracies of the base classifiers. Many researchers have demonstrated effectiveness of this combining algorithm and its modifications [2, 4, 6, 15].

3 Wrapper for Ensemble Based on Contextual Features

In this section we consider our wrapper approach based technique to construct a diverse ensemble of classifiers using the CM measure [1, 7]. First we describe the ensemble construction and then introduce the enhanced Contextual Ensemble Feature Selection algorithm (CEFS), which implements our approach.

Ensemble construction process is performed during the learning phase and it includes the base classifiers learning using the training set. The general process of the base classifier generation can be described as it is shown in Figure 1.

First, an empty ensemble is set (block *1*), then a heuristic is defined (block *2*) for the classifier generation (block *3*). The input for the classifier generation is a modified training set $\mathbf{T}_i$ including a sample of the training instances and set of features. With this input a new classifier is generated (block *4*) and evaluated (block *5*). Subsequent modification of the ensemble (block *6*) includes addition and/or deletion of some classifier from/to the ensemble. In blocks *7* and *8*, some ensemble characteristics are estimated and they are used for stopping criterion evaluation. Block *9* is a decision block that can alter the process of next classifier generation (dashed arrow to the block *3*), modify the heuristic rule (block *10*), or stop the process with the final ensemble of base classifiers (block *11*).

Real construction techniques do not necessarily include all these components. There exist three major ways to generate base classifiers. The first one is to generate a set of base classifiers without their evaluation according to a given heuristic, as in bagging [3]. In this case blocks *5*, *7*, and *10* with the corresponding output are not in use. The second approach changes the rule of generation each time when a new classifier is added to the ensemble, as in Boosting [20]. In this case blocks *5* and *7* are not present. In these two cases the stopping criterion is defined as the cycle counter. The third approach uses the cycle for the ensemble refinement with the stopping criterion, as with neural network classifiers in [13]. The latter way is particularly interesting because feature selection with a heuristic rule can be used in different ways to guide the generation of a good ensemble. The techniques, in which some ensemble characteristics are used for the new classifier evaluation, are called *wrapper* techniques and are considered for example in [8, 9, 10].

Now we present the outline of our enhanced algorithm called CEFS (Contextual Ensemble Feature Selection). Preliminary version of this algorithm was presented in

[21]. CEFS constructs base classifiers (one for each class) for a training set over selected feature subsets that are formed using the CM value.

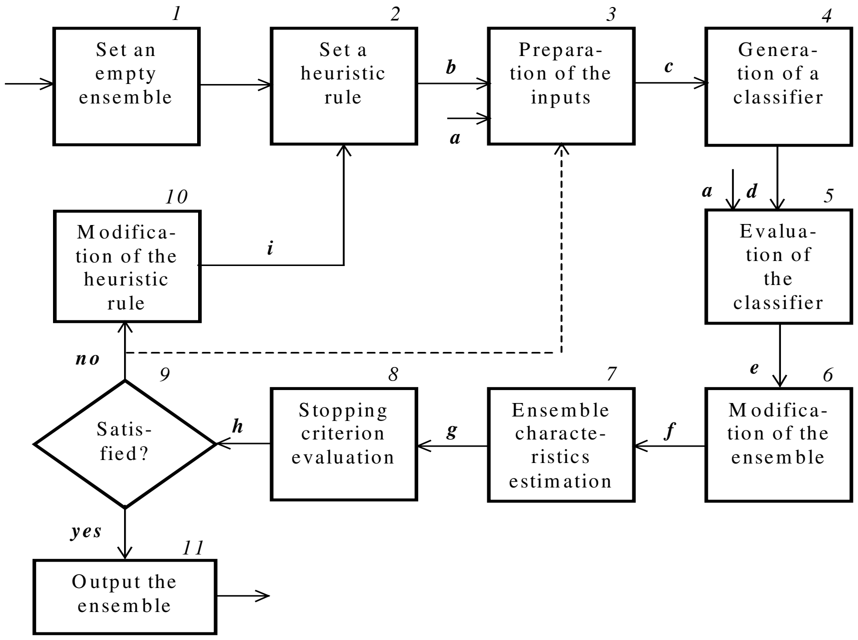

1
Set an empty ensemble
2
Set a heuristic rule
3
Prepara-tion of the inputs
4
Genera-tion of a classifier
5
Evalua-tion of the classifier
6
Modifica-tion of the ensemble
7
Ensemble characte-ristics estimation
8
Stopping criterion evaluation
9
Satis-fied?
10
Modifica-tion of the heuristic rule
11
Output the ensemble
a
b
c
d
e
f
g
h
i
no
yes

Fig. 1. The scheme of base classifiers' generation. Block symbols represent the basic process components, and small letters associated with arrows denote the input/output data for each component. Small letters near the arrows denote the following: ***a*** – training set **T**, ***b*** – heuristic rule, ***c*** – inputs for the classifier generation, ***d*** – classifier, ***e*** – fitness value for classifier, ***f*** – current set of classifiers in the ensemble, ***g*** – control ensemble characteristics, ***h*** – control value for the stopping criterion, ***i*** – parameters for the heuristic rule.

The objective is to construct an ensemble of classifiers so that each classifier is as accurate as possible for one class with respect to the other classes. Each base classifier of the ensemble is based on a fixed number of features with the highest CM value. First, the initial ensemble is constructed on the same fixed number of features for all the classifiers of the ensemble. Then, the ensemble is iteratively modified trying to change the fixed number of features suggesting exclusions or inclusions of some features. This is guided by the diversity of classifiers and the CM value.

The diversity of an ensemble is suggested as a measure of independence of the base classifiers [17]. We modify the formula to be able to calculate the diversity Div_i of a classifier i which can be approximated using the formula (1):

$$Div_i = \frac{\sum_{j=1}^{M} \sum_{k=1,k\neq i}^{|S|} Dif\left(h_i(x_j), h_k(x_j)\right)}{M \cdot \left(|S|-1\right)} \qquad (1)$$

where $|S|$ denotes the number of the base classifiers, $h_i(\mathbf{x}_j)$ denotes the classification of the instance $\mathbf{x}_j$ by the classifier h_i, and $Dif(a,b)$ is zero if the classifications a and b are the same and one if they are different, and M is the number of instances in the test set.

We use the CM measure as it has been presented in [7] and described below. Let the difference $d_{rs}^{(f_i)}$ between the values of a categorical feature f_i for the instances r and s be 0, if the values are same and 1, otherwise. Between the values of a continuous feature f_i the difference is correspondingly calculated by $d_{rs}^{(f_i)} = \min\left(\left|f_i(r) - f_i(s)\right| / t_{f_i}, 1\right)$, where $f_i(\mathbf{x})$ is the value of the feature f_i in the instance $\mathbf{x}$, and t_{f_i} is a threshold. In this paper it is selected to be one-half of the range of the values of the feature f_i.

Then the distance D_{rs} between the instances r and s is $D_{rs} = \sum_{i=1}^{N} d_{rs}^{f_i}$, where N is the number of the features. The value of CM measure CM_{fi} of a feature f_i is defined to be

$$CM_{f_i} = \sum_{r=1}^{M} \sum_{s \in \overline{C(r)}} w_{rs}^{(f_i)} d_{rs}^{(f_i)} \qquad \textbf{(2)}$$

where M is the number of instances, $\overline{C(r)}$ is the set of instances not from the same class as the instance r, and $w_{rs}^{(f_i)}$ is a weight chosen so that instances that are close to each other, i.e., that differ only in a few features, have greater influence in determining each feature's CM measure value. In [10] weights $w_{rs}^{(f_i)} = 1/D^2_{rs}$ were used when s is one of the k nearest neighbors of r, in terms of D_{rs}, in the set $\overline{C(r)}$, and $w^{(fi)}_{rs} = 0$ otherwise. The number of nearest neighbors k used in [7] was the binary logarithm of the number of examples in the set $\overline{C(r)}$.

In this paper we calculate the values of the CM measure for different classes and call them FMERITS in the algorithm. Thus, in the external loop of the formula (2) instead of all M instances of the set, only instances of a particular class are considered. This gives us a possibility to define importance of features for distinguishing one class from the others, and to build an ensemble of classifiers based on features important for some class in multi-class problems.

Our algorithm is composed of two main phases: 1) the construction of the initial ensemble and 2) the iterative development of new candidate ensembles. The main outline of our algorithm is presented in Figure 2.

In the CEFS algorithm ensembles are generated in two places, the initial ensemble is generated using a different procedure than the generation of the candidate ensembles that is included in the procedure *cycle*. In the generation of the initial ensemble we use the approach presented in [21] that is an adjusted version of the approach in [16]. The iteration mechanism used in this paper changes one base classifier in each iteration. The base classifier with the smallest diversity is tried to be changed. When the classifier to be changed has been selected, one feature is tried to be added or deleted from the subset of features that was used to train the classifier. The feature is selected using the value of the CM measure.

4 Experiments

In this Section, experiments with our algorithm for generation of an ensemble of classifiers built on different feature subsets are presented. First, the experimental setting is described, and then, results of the experiments are presented. The experiments are conducted on seven data sets taken from the UCI machine learning repository [12]. The main characteristics of the seven data sets are presented in Table 1. The table includes the name of the data set, the number of instances included in the data set, the number of different classes of instances, and the numbers of different kind of features included in the instances.

```
Algorithm CEFS(DS)

DS       the whole data set
TS       training set
VS       validation set used during the refinement cycle
TS       final evaluation set
Ccurr    the current ensemble of base classifiers
accu     the accuracy of the current ensemble
FS       set of feature subsets for the base classifiers
FMERITS  set of CM feature merits for each class

begin
   divide_instances(DS,TS,VS,TS) {divides DS into TS,
   VS, and TS using stratified random sampling}
   Ccurr=build_initial_ensemble(TS,FS,FMERITS)
   accu=accuracy(VS,Ccurr) {calculates the accuracy of
   the current ensemble over the test set}
   loop
      cycle(TS,Ccurr,FS,FMERITS,accu){developes candi
      date ensembles and updates accu, Ccurr, and FS
      when the current ensemble is changed}
   until no_changes
   accu=accuracy(TS,Ccurr)
end algorithm CEFS
```

Fig. 2. Outline of the CEFS algorithm

For each data set 30 test runs are made. Each time 60 percent of the instances are picked up to the training set and the rest of the instances of the data set are divided into two approximately equal sets (VS and TS). The first set, VS is used to calculate the initial accuracies and for tuning the ensemble of classifiers, adjusting the initial feature subsets so that the ensemble accuracy becomes as high as possible using the selected heuristics. The other set, TS is used for the final estimation of the accuracy.

Table 1. Data sets used in the experiments

Data set	Instances	Classes	Features	
			Discrete	Continuous
Car	1728	4	0	5
Glass	214	6	0	9
Iris	150	3	0	4
Lymphography	148	4	15	3
Thyroid	215	3	0	5
Vehicle	846	4	0	18
Zoo	101	7	16	0

The use of single test set only would give optimistically biased estimation of the ensemble accuracy, as can be seen also from the experiments. At each run of the refinement cycle, besides the classification accuracies of the base classifiers and the ensemble, we collect the numbers of features being deleted and added during the cycles.

The base classifiers themselves are learnt using the C4.5 decision tree algorithm with pruning [18]. The test environment was implemented within the MLC++ framework (the machine learning library in C++) [11].

We first made test runs with different initial amounts of features for the seven data sets. The number of features was fixed by using CM values related threshold value. First, the features are ordered in descending order according to their CM values and the threshold value is used as portion of the interval between the lowest and highest CM value. The average selected amounts of the features for the threshold value 0.9 were actually between 62% and 87% of all the features of the data sets and for the threshold value 0.1 correspondingly between 9% and 34%. The actual amounts of features for each best initial ensemble were (the threshold value in parenthesis): Lymphography 36% (0.5), Vehicle 69% (0.9), Thyroid 62% (0.7), Zoo 63% (0.9), Car 84% (0.9), Iris 38% (0.3), and Glass 51% (0.5). The accuracy values for the best initial ensembles are included in Figure 3 as *initacc* values. As can be seen the accuracies of the initial ensembles are higher than the accuracies achieved using C4.5 with all features for Lymphography, Thyroid, Iris, and Glass data sets and lower for Vehicle, Zoo, and Car data sets.

These results also support the finding that in some cases ensembles of classifiers built on different feature subsets are useful. Because we used the CM values to select the features, the above results may help us in organizing experiments with the refinement cycle. The amount of features in the optimal situation may be different from the above results if the CM value is biased so that it suggests including features in a wrong order. Anyway we use these best initial ensembles in our GEFS algorithm as the initial ensembles.

The results of test runs with refinement cycle for the seven data sets are also presented in Figure 3. The accuracies of the last ensembles at the end of iteration are presented as *finalacc* values. The number of the base classifiers is kept fixed being equal to the amount of classes in the data set. As can be seen, for all data sets the accuracy of the last ensemble is higher than with the first ensemble, except Thyroid data set. For the data sets Iris and Thyroid the change in the accuracy remained very small.

It is also seen that the accuracies of the final ensembles with weighted voting (wv) are always higher than the accuracies achieved using C4.5 (C4.5) for the test set TS. This shows that the refinement cycle with the CM and diversity values used in the algorithm are able to raise the accuracies with these data sets.

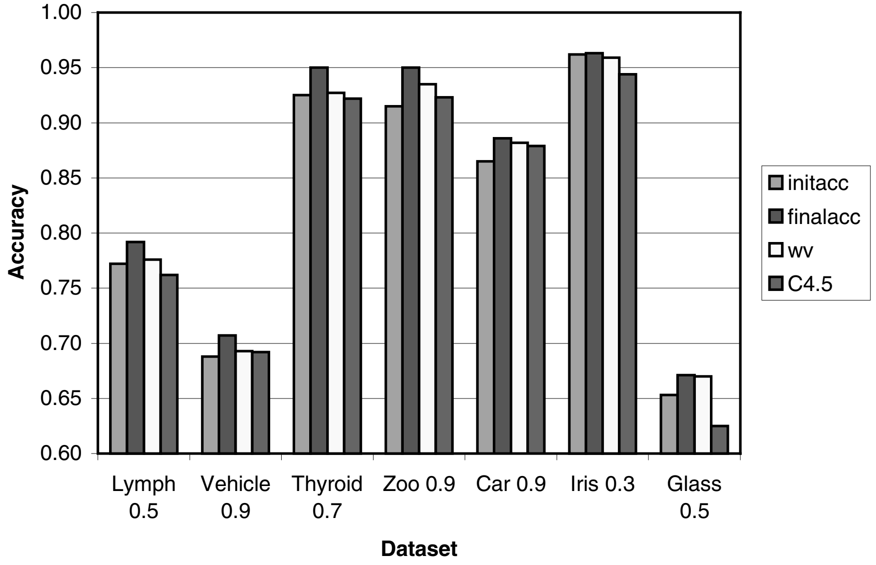


Fig. 3. Accuracies of the ensembles before (initacc) and after (finalacc) the refinement cycle for VS instances and for WV and C4.5 for TS instances

The average numbers of features in the initial and final ensembles are presented in Figure 4. Also the total number of the features in each data set is included in Figure 4. As can be seen with most data sets the refinement adds some features. As supposed only subset of features are used and still the ensemble accuracies are higher. The percentages of features included in the final ensemble are: Lymphography 47%, Vehicle 70%, Thyroid 61%, Zoo 46%, Car 85%, Iris 68%, and Glass 48%.

5 Conclusions

Ensembles of classifiers can be constructed by a number of methods manipulating the training set with the purpose of creating a set of diverse and accurate base classifiers. In this paper, we have proposed an approach that builds an ensemble of classifiers on different feature subsets. Each base classifier in the ensemble is built on features with the highest Contextual Merits to distinguish one class from the others. We also proposed the refinement of ensembles. Our refinement cycle is defined in terms of wrapper approach where the diversity and classification accuracy of the ensemble serve as an evaluation function and a stopping criterion accordingly. In each cycle of the ensemble refinement procedure we try to change the feature subset of the base classifier with minimum diversity value to raise the ensemble diversity and thus classification accuracy.

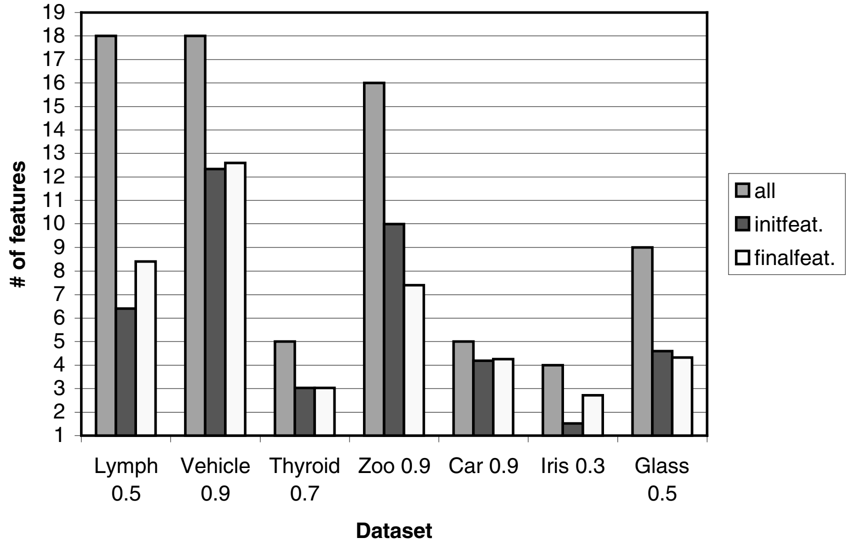


Fig. 4. Number of features in the data set (all), average number of features in the initial (initfeat.) and final ensemble (finalfeat.)

We have evaluated our approach on a number of data sets from the UCI machine learning repository. The experimental results supported the finding that in some cases ensembles of classifiers built on different feature subsets are better than single classifiers built on the whole feature set.

Estimation of the particular domain characteristics, which will determine appropriateness of the CM measure as a heuristic rule for feature selection in ensembles, is an important question for further research. Improvement of the ensemble refinement cycle is also an interesting topic for future research.

Acknowledgements. This research is partially supported by the Academy of Finland (project number 52599), Center for the International Mobility (CIMO) and the COMAS Graduate School of the University of Jyväskylä. We would like to thank the UCI machine learning repository, domain theories and data generators for the data sets, and the machine learning library in C++ for the source code used in this study.

References

1. Apte, C., Hong, S.J., Hosking, J.R.M., Lepre, J., Pednault, E.P.D., Rosen, B.K.: Decomposition of Heterogeneous Classification Problems. Advances in Intelligent Data Analysis, Springer-Verlag, London (1997) 17-28.

2. Batitti, R., Colla, A.M.: Democracy in Neural Nets: Voting Schemes for Classification. Neural Networks, Vol. 7, No. 4 (1994) 691-707.
3. Brieman, L., Friedman, J., Olshen, R., Stone, C.: Classification and regression trees. Wadsworth International Group, Belmont, California (1984).
4. Cost, S., Salzberg, S.: A Weighted Nearest Neighbor Algorithm for Learning with Symbolic Features. Machine Learning, Vol. 10, No. 1 (1993) 57-78.
5. Dietterich, T. Machine Learning research: Four Current Directions. Artificial Intelligence, Vol. 18, No. 4 (1997) 97-136.
6. Hansen, L., Salamon, P.: Neural Network Ensembles. In: IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 12 (1990) 993-1001.
7. Hong, S.J.: Use of contextual information for feature ranking and discretization. IEEE Transactions on knowledge and Data Engineering, Vol. 9, No. 5) (1997) 718-730.
8. John, G.H.: Enhancements to the Data Mining Process, PhD Thesis, Computer Science Department, School of Engineering, Stanford University (1997).
9. Kohavi R., John, G.H.: Wrappers for feature subset selection. Artificial Intelligence Journal, Special Issue on Relevance edited by R. Greiner, J. Pearl and D. Subramanian.
10. Kohavi, R., John, G.H.: The Wrapper Approach. In: (eds.) H. Liu and H. Motoda, Feature Selection for Knowledge Discovery in Databases, Springer-Verlag (1998).
11. Kohavi, R., Sommerfield, D., Dougherty, J.: Data Mining Using MLC++: A Machine Learning Library in C++. Tools with Artificial Intelligence, IEEE CS Press (1996) 234-245.
12. Merz, C.J., Murphy, P.M.: UCI Repository of Machine Learning Datasets [http://www.ics.uci.edu/ ~mlearn/MLRepository.html]. Dept of Information and CS, Un-ty of California, Irvine, CA (1998).
13. Opitz, D. Feature Selection for Ensembles. In: 16th National Conf. on Artificial Intelligence (AAAI), Orlando, Florida (1999) 379-384.
14. Opitz, D., Maclin, R.: Popular Ensemble Methods: An Empirical Study. Artificial Intelligent Research, Vol. 11 (1999), 169-198.
15. Opitz, D., Shavlik, J.: Generating accurate and diverse members of neural network ensemble. Advances in Neural Information Processing Systems, Vol. 8 (1996) 881-887.
16. Oza, N., Tumer, K.: Dimensionality Reduction Through Classifier Ensembles. Tech. Rep. NASA-ARC-IC-1999-126.
17. Prodromidis, A. L., Stolfo, S. J., Chan P. K.: Puning Classifiers in a Distributed Meta-Learning System. In: Proc. of 1st National Conference on New Information Technologies, (1998) 151-160.
18. Quinlan, J.R.: C4.5: programs for machine learning. Morgan Kaufmann, San Mateo, California (1993).
19. Shapire, R.E., Freud, Y., Bartlett, P., Lee, W.S.: Boosting the Margin: A New Explanation of the Effectiveness of the Voting Methods. The Annals of Statistics, Vol. 25, No. 5 (1998), 1651-1686.
20. Shapire, R.E.: A Brief Introduction to Boosting. In: Proceedings of 16th International Joint Conference on Artificial Intelligence (1999).
21. Skrypnyk, I., Puuronen, S.: Ensembles of Classifiers based on Contextual Features. In: Proceedings of 4th International Conference “New Information Technologies” (NITe’2000), Minsk, Belarus, Dec. (2000) (to appear).

Interactive Clustering for Transaction Data ⋆

Yongqiao Xiao, Margaret H. Dunham
Department of Computer Science and Engineering
Southern Methodist University, Dallas, Texas 75275-0122
{xiao,mhd}@seas.smu.edu

Abstract. We propose a clustering algorithm, OAK, targeted to transaction data as typified by market basket data, web documents, and categorical data. OAK is interactive, incremental, and scalable. Use of a dendrogram facilitates the dynamic modification of the number of clusters. In addition, a condensation technique ensures that the dendrogram (regardless of database size) can be memory resident. A performance study shows that the quality of clusters is comparable to ROCK[7] with reduced complexity.

1 Introduction

Clustering is one of the major data mining problems [11]. Its objective is to classify data from a given dataset into groups such that the items within a group are more similar to each other than to those outside the group. Traditional clustering techniques target small databases and are performed in batch algorithms. A user provides the dataset and may indicate the number of desired clusters, and the algorithm presents the results to the user. Recent research has examined the need for *on-line (interactive)* data mining algorithms [10]. The purpose of on-line algorithms are to provide continuous feedback to the user and to allow the whole process to be controllable by the user. (We prefer the term interactive as on-line connotes some sort of real time processing.)

In this paper we propose a new clustering algorithm, OAK, targeted to the domain of transaction data. The most common example of transaction data is market basket data. Associated with each purchase is a cash register receipt that lists the data purchased. Each of these is called a transaction and it contains a subset of the total set of items which could be purchased. Transaction data actually occurs in many different application domains. In Information Retrieval, a document can be summarized by a set of keywords associated with it. Thus Web search engines could represent documents (and even user queries) as types of transactions. Categorical data can be viewed as a special type of transaction data, [7], where each record containing categorical data is a transaction. It is shown in [7] that the use of traditional distance measures, such as Euclidean, may not work well with categorical data.

⋆ This material is based upon work supported by the National Science Foundation under Grant No. IIS-9820841

Some transaction data, for example, the results of a web search engine, may not be provided all at once to the clustering algorithm since the data are coming continuously and possibly slowly through the network. Showing the current clusters dynamically as the data comes and the algorithm runs, will help bridge the network gap and give the user a rough idea about what the final clusters would be like. This could also allow the user to adjust the number of clusters. To support this type of processing, then, the following are desirable features for any effective transaction data clustering algorithm:

1. **Transaction Data** - The algorithm should be targeted to efficiently cluster transaction data. Techniques designed for numeric data may not perform effectively with transaction data.
2. **Incremental** - The cluster should be performed in an incremental mode as the data is available. The entire database may not be available at one time.
3. **Interactive** - As the data is being clustered, the user should be provided feedback about the status of the clustering process. This on-line feature allows the user to change any input parameters desired, such as the number of clusters, k. When the data is not known apriori, choosing an appropriate value of k may be difficult.
4. **Scalability** - The size of the transaction database may be quite large. Any algorithm should be able to function even with limited main memory.

The remainder of the paper is outlined as follows: Section 2 reviews related work. Section 3 sketches the algorithm and highlights it major features. The two kinds of user interactions are described in Section 4. The adaptability to large databases and data input order is described in Section 5. Section 6 reports on the experimental results, and Section 7 concludes the paper.

2 Related Work

Conventional clustering algorithms can be classified into *partitional clustering* and *hierarchical clustering*[11]. With hierarchical approaches a nested set of clusters is created, while with partitional only one is. Hierarchical agglomerative algorithms create the clusters in a bottom up fashion starting with each point in its own cluster and then successively merging clusters until either the desired number of clusters is obtained or only one cluster exists. A tree or *dendrogram* is used to show the structure of the clusters. Agglomerative hierarchical clustering algorithms differ in the similarity/distance measures used to merge clusters.

Many clustering algorithms for large databases have recently been proposed [17,1,3,7,6]. Among them, BIRCH[17] and ScaleKM[3] compress the large database into subclusters, ROCK[7] uses sampling and labeling techniques. CLIQUE[1] proposes to find clusters embedded in subspaces of high dimensional data.

There have also been several clustering algorithms targeted to categorical data. ROCK proposes a novel similarity measure between points based on the number of common neighbors, called *links* [7]. CACTUS[6] introduces a formalization of clusters for categorical attributes and finds clusters using inter-attribute and intra-attribute summaries.

Clustering of market basket data was examined in [8]. In this case a weighted hypergraph is created from large itemsets needed to generate association rules, and clustering is accomplished via a hypergraph partitioning algorithm. It has been pointed out that this approach can lead to poor clustering results [7]. The assignment of a transaction, t_i, to a cluster, C_j, can be determined using a similarity measure of $\frac{|t_i \cap C_j|}{|C_j|}$.

Recently clustering has been applied to the retrieval results of Web search engines [9, 16]. Scatter/Gather [9] applies clustering to the entire document collection and allows interactive browsing of the clusters. STC[16] uses a suffix tree to identify sets of documents that share common phrases.

3 Overview of OAK

In this section we provide an overview of our new clustering algorithm. A more detailed discussion is available [15]. Our proposed algorithm, *OAK(Online Adaptive Klustering)*, does satisfy the desirable properties identified in Section 1. The Cosine Similarity[1] measure used in Information Retrieval[13] is adopted to measure the similarity between transactions. An agglomerative hierarchical clustering algorithm is used and a dendrogram is maintained which facilitates the user dynamically changing the number of desired clusters. Any such algorithm can be used as long as the clustering is incremental. Scalability and limited main memory are handled by providing a method to condense the dendrogram.

The dendrogram which is created by OAK actually may not have one leaf node per point. Instead a maximum number of representatives are used as the leaf nodes. Here a *representative* represents an individual point or a set of similar points(subcluster). If the database is small there may be one representative per point to be clustered (as with a conventional dendrogram). For larger databases, each representative would take the place of several points. While the hierarchical clustering occurs, if main memory is determined to be insufficient to store the complete dendrogram, part of it will be condensed into a new representative leaf. This dynamic condensation approach, then, ensures that the dendrogram will fit into main memory regardless of the memory size.

The algorithm is shown in Algorithm 1. The overall time complexity is $O(MN)$, where M is the number of representatives and N is the number of data points in the database. The updating of the dendrogram with inclusion of each new point requires $O(M)$[14]. The updating of cluster profiles and the condensation also have $O(M)$ time complexity for each new point, which are described in the following sections. We use SLINK[14] to incrementally update the dendrogram in the paper, but any incremental agglomerative hierarchical algorithm such as CLINK[5] could be used.

Algorithm 1 *OAK*

[1] $Cos(p_i, p_j) = \frac{<p_i, p_j>}{||p_i|| ||p_j||}$, where " $<,>$ " denotes inner product, and "$||.||$" denotes vector norm

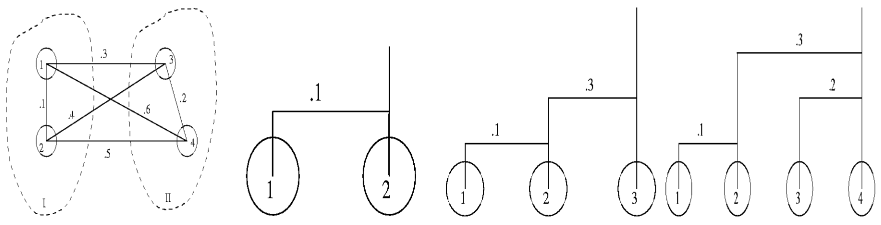
```
Input:
    $t_1, t_2, \ldots, t_N$: sequence of data points(transactions).
    $k$: number of clusters.
    $M$: number of representatives; May be determined by memory size.
Output:
    the clusters and their profiles.
Method:
        //global data structure definitions.
        $rep[1..M]$: vectors for the representatives.
        $\lambda[1..M]$: dissimilarity at each level of the dendrogram.
        $\pi[1..M]$: the merging pair at each level of the dendrogram.
        $d_{min}$: threshold for condensation.
        $profile[1..k]$: profile of each cluster.
(1)     $numRep = 1$;
(2)     $rep[1] = t_1$;
(3)     $\pi[1] = 1$; $\lambda[1] = \infty$;
(4)     for $i$ from 2 to $N$ do begin
(5)         read $t_i$.
(6)         if $i \leq M$ then
(7)             update $\lambda$ and $\pi$ incrementally, e.g. by SLINK.
(8)             numRep++;
(9)             $rep[numRep] = t_i$;
(10)            $d_{min} = min\{\lambda[j] | j \in [1..numRep]\}$;
(11)            UpdateProfile(k, numRep);
(12)        else
                    //condense the most similar pair or remove an outlier.
(13)            Condense(k, numRep, $t_i$);
(14)        endif
(15)        if interrupted by the user
(16)            k=InteractWithUser(k, numRep);//a new parameter may be chosen
(17)    end
```

The dendrogram is efficiently represented by the pointer representation[14]. A pointer representation consists of two functions: $\pi : 1, \ldots, N \rightarrow 1, \ldots, N$, and $\lambda : 1, \ldots, N \rightarrow [0, \infty]$, where π and λ satisfy the following conditions:

$$\pi(N) = N, \quad \pi(i) > i, for\ i < N.$$
$$\lambda(N) = \infty, \quad \lambda(\pi(i)) > \lambda(i), for\ i < N.$$

Basically, $\lambda(i)$ is the lowest level at which i is no longer the last point in its cluster, say x, and $\pi(i)$ is the last point in the cluster x[14]. It is assumed that the points are identified by the integers $1, 2, \cdots, N$.

Example 1. Suppose that a database consists of points $\{1, 2, 3, 4\}$ and the points are read in the same order. Figure 1a) shows the distances between the four



a)A Weighted Graph b)Two Points c)Three Points d)Four Points

Fig. 1. Graph and Cluster Hierarchies

points. Figures 1b), c) and d) show the successive dendrograms after new points are clustered by single-linkage clustering. Note that a level of a dendrogram is represented by the distance value at that level, assuming all distance values are unique.

3.1 Updating of Cluster Profiles

User interaction is peformed via a cluster profile. A *cluster profile* is a summary of a cluster's contents, which may include the number of points, the most frequent items, or even association rules in the cluster. In our discussions we assume that the profile contains at least the number of points in a cluster. The user can interact at any point during the clustering process. For each user interaction, the algorithm shows the up-to-date cluster profiles to the user, and if the user changes the number of clusters, the profiles of the new clusters are updated and shown to the user.

There are three cases with respect to the changing of the clusters after a new point is clustered. The cluster profiles are updated accordingly. Due to space, the details are not shown here, but in [15].

4 Interactive Clustering

The number of clusters to be created can be changed dynamically and interactively by the user. It can be done in an ad-hoc or in a browsing fashion (e.g., roll-up or drill-down of the dendrogram), which we call *ad-hoc interaction* and *browsing interaction*, respectively.

4.1 Ad-Hoc Interaction

With ad-hoc interactions, the user indicates a new value for k, independent of the current value. It is crucial to efficiently compute the new clusters for any new k. To do this, we need to determine the cluster to which each point belongs, which we call cluster representation. A cluster representation is related to the σ function[14], which is defined as:

$$\sigma(i,h) = min\{\pi^j(i)|\lambda(\pi^j(i)) \geq {}^2h, j \geq 0\}.$$

where $\pi^0(i) = i, \pi^2(i) = \pi(\pi(i))$ and π^j repeats π j times, and the sequence $\pi^j(i)(j \geq 0)$ is finite with the last element being the current number of representatives(let it be $m \leq M$). The function $\sigma(i,h)$ gives the cluster of point i at level h. Given the number of clusters k, level h is the kth largest λ value. Finding the kth largest value among m numbers is a selection problem, which can be done linearly as shown in [4]. Since k is usually much smaller than m, another possible way to find h is to first build a heap with the m values and then extract the largest, the second largest, until the kth largest. The total time is $O(m + k \times log(m))$, where $O(m)$ is to build the heap bottom up, and $O(log(m))$ for each extraction of the largest value from the heap. Note that if there are duplicate values in λ, then the pairs with the same dissimilarity are regarded as being merged at the same level.

Based on Lemma 1, we develop a linear method to get the cluster representation. The function is not shown here due to space, but in [15].

Lemma 1. *Let m be the number of representatives, λ, π be the pointer representation for the m representatives, k be the current number of clusters set by the user, C be the cluster representation. For any point $i(1 \leq i \leq m)$, we have:*

$$C(i) = C(\pi(i)) = \cdots = C(\pi^{j_{min}}(i)),$$

where $j_{min} = min\{j|j \geq 0, \lambda(\pi^j(i)) \geq h\}$, where h is the kth largest value of λ.
Proof: *See [15].*

4.2 Browsing Interaction

With browsing interactions, the parameter k can be changed to $k+1$ or $k-1$. It is shown in [15] that using the heap data structure allows efficient such interactions.

5 Adaptive Clustering

The adaptability to large databases is achieved by a dynamic condensation technique, and sensitivity to data input order is lessened by a conservative outlier removal policy. The Condense function can be found in [15] due to space.

5.1 Dynamic Linear Condensation

Since the most similar pair lies at the lowest level of the dendrogram, their dissimilarity(let it be d_{min}) is used as a threshold for condensation. There are two types of condensations. If the dissimilarity between the new point and its nearest

[2] " $>$ " was used in [14], we change for the convenience of the following description

neighbor is less than d_{min}, then they are condensed into one representative, which we call type I condensation. Type II condensation happens if d_{min} is smaller. In this case the two points with dissimilarity d_{min} are condensed into one representative, and then d_{min} is set to the next level in the new dendrogram with the new point clustered.

Both types of condensations always condense the most similar pairs, referred to as *reciprocal nearest neighbors(RNNs)* [12]. Condensing the RNNs has a nice property, called the *Reducibility Property* [12]: The new dissimilarity between the condensed RNNs and the other representatives is nondecreasing for linkage clustering methods. This property ensures that after condensation the existing dendrogram is not changed. Therefore, the cluster profiles with condensation can be efficiently updated. For type I condensation, we only need to increment the profile of the cluster to which the new point is condensed. For type II condensation, we need to cluster the new point into the condensed hierarchy, and the new cluster profiles are updated in the same way as described in the above section.

For efficiency, the two condensed points/subclusters are represented by their centroid, which can be computed from the sums of the two vectors for the two points. The centroid is then regarded as a point(representative) and used to compute the dissimilarity with the new points not yet clustered.

5.2 Outlier Detection and Removal

To be less sensitive to data input order, our outlier detection is conservative: (1) only a point that has stayed in the dendrogram for some minimum time(t_{min}) is the point detected as an outlier. (2) once an outlier is detected, it is not thrown away. Instead it can be put in a main memory buffer or be written to hard disk if no available main memory. After all data points are read the outliers are rescanned and reclustered as new data points. Reclustering the outliers can recover those that might be wrongly detected as outliers. More details can be found in [15].

6 Experimental Results

We report on two different sets of experiments. Two real-life data sets are used to compare the quality of clustering of our algorithm OAK with ROCK[7]. We also examine the performance of OAK using two large synthetic data sets to demonstrate the adaptability of OAK through dynamic condensation and its insensitivity to data input order. In all cases, we focus on the quality of clustering as the measure. All experiments were conducted on a Compaq PC with a 500MHz Pentium III processor and 128 MB of RAM and running Microsoft Windows 98. OAK was implemented in Java, and was executed by the Java Virtual Machine.

6.1 Results with Real-life Data Sets

The Congressional votes and Mushroom [3] in [7] are used in our experiments. The Congressional voting data set is the United States Congressional Voting Records in 1984. Each record corresponds to one Congressman's Votes on 16 issues. The mushroom data set is derived from *The Audubon Society Field Guide to North American Mushrooms.* Each record contains information that describes the physical characteristics of a single mushroom.

Cluster Profiles We first show the clustering results when there are no condensations. M is set to 435 and 8124 for the Congressional voting and mushroom data, respectively. Outliers are removed at the end of clustering by user interactions. That is, we set a larger k and select the largest clusters among the k clusters as output.

For the Congressional voting data, we first set k to 2. From the cluster profiles we found that one cluster has only one data point. We knew that there must be outliers, and then increased k until we found two big clusters in the cluster profiles. When $k = 55$, OAK had a slightly better result than ROCK. The details about the result can be found in [15].

For the mushroom data set, when k is set to 21 and all clusters are selected as output, OAK had exactly the same result as ROCK (not shown due to space), i.e., only one small cluster is mixed with poisonous and edible, and the others are either poisonous or edible. More details can be found in [15].

Effect of Condensation We tested four cases: $M = 1000$, $M = 500$, $M = 200$ and $M = 100$ for the mushroom data set. The maximum condensation threshold is set to $d_{max} = 0.5$, and was never exceeded for the four cases. In order to compare with the result without condensation, outliers are removed only at the end of clustering by user interactions.

The five big clusters are selected as output from $k = 21$ clusters for all cases. With $M = 1000$ and $M = 500$, we still got 100% accuracy for each of the five big clusters. With $M = 200$ and $M = 100$, four of the five clusters still had 100% accuracy. The fourth cluster is mixed up with both poisonous and edible mushrooms, with the majority been edible mushrooms. While keeping high accuracy, OAK also speeds up linearly with M decreasing. The execution time of OAK with the above different M's for the Mushroom data set is shown in Figure 2. It can be seen that the execution time is roughly linear to the number of representatives except that when $M = 8124$ the execution time is below linear due to the fact that there is no condensation overhead.

6.2 Results with Synthetic Data Sets

We use two synthetic data sets, which are market basket databases. The total number of transactions is 100K. The two data sets have exactly the same

[3] Obtained from UCI Machine Learning Repository (http://www.ics.uci.edu/~mlearn/MLRepository.html.

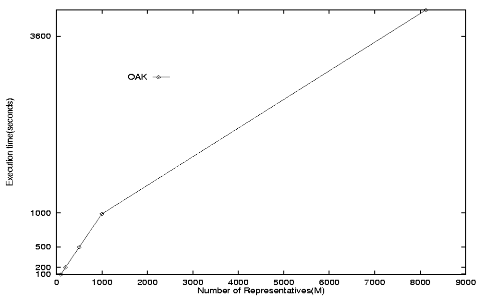


Fig. 2. Execution time(by Java) of OAK with M varying for the Mushroom data set

transactions but differ in the order of transactions. One is skewed(one cluster concatenated after another), and the other is randomly distributed(transactions in each cluster are redistributed randomly). OAK found the two clusters with small errors for the two data sets, which demonstrates that OAK is less sensitive to the input order. The details can be found in [15].

6.3 Performance Summary

We have found that OAK has a comparable (if not slightly better) quality of clustering when compared to ROCK. It accomplishes this with a reduction from cubic to quadratic in time complexity and from quadratic to linear in space. Condensation has been shown to be effective with the overall impact being a linear reduction in time as the number of representatives decreases.

7 Conclusion and Future Work

We presented an interactive clustering algorithm for transaction databases. User interactions are achieved by showing the user the profiles of the current clusters and allowing the user to adjust the number of clusters during the clustering process. Large databases are condensed into a much smaller number of representatives through a dynamic condensation technique, which is shown to be effective and efficient. Sensitivity to data input order is lessened by a conservative outlier removal policy.

Future work includes: (1) automatically identifying the number of clusters for output at the end of clustering, which are now obtained by user interactions; (2) dynamically determining the number of representatives, which is now fixed as an input parameter; (3) enhancing cluster profiles with association rules [2] for market basket data;

References

1. Rakesh Agrawal, Johannes Gehrke, Dimitrios Gunopulos, and Prabhakar Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, pages 94–105, 1998.
2. Rakesh Agrawal and Ramakrishnan Srikant. Fast Algorithms for Mining Association Rules in Large Databases. In *Proceedings of the Twentieth International Conference on Very Large Databases*, pages 487–499, Santiago, Chile, 1994.
3. Paul Bradlay, Usama Fayyad, and Cory Reina. Scaling clustering algortihms to large databases. In *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining (KDD-98)*, pages 9–15, 1998.
4. Thomas H. Cormen, Charles E. Lerserson, and Ronald L. Rivest. *Introduction to Algorithms*. The MIT Press, Cambridge, MA, 1990.
5. D. Defays. An efficient algorithm for a complete link method. *The Computer Journal*, 20(4):364–366, 1977.
6. Venkatesh Ganti, Johannes Gehrke, and Raghu Ramakrishnan. CACTUS - clustering categorical data using summaries. In *Proceedings of the 5th International Conference on Knowledge Discovery and Data Mining (KDD-99)*, pages 73–83, 1999.
7. Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. ROCK: A robust clustering algorithm for categorical attributes. In *Proceedings of the 1999 IEEE International Conference on Data Engineering*, pages 512–521, 1999.
8. Eui-Hong Han, George Karypis, Vipin Kumar, and Bamshad Mobasher. Clustering based on association rule hypergraphs. In *1997 SIGMOD Workshop on Researhc Issues on Data Mining and Knowledge Discovery*, 1997.
9. Marti A. Hearst and Jan O. Pedersen. Reexamining the cluster hypothesis: Scatter/gather on retrieval results. In *Proceedings of the 19th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 76–84, 1996.
10. Christian Hidber. Online association rule mining. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, pages 145–156, 1999.
11. Anil K. Jain and Richard C. Dubes. *Algorithms for Clustering Data*. Prentice Hall, Englewood Cliffs, New Jersey, 1988.
12. F. Murtagh. A survey of recent advances in hierarchical clustering algorithms. *The Computer Journal*, 26(4):354–359, 1983.
13. G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill Book Co., McGraw-Hill, New York, 1983.
14. R. Sibson. Slink: An optimally efficient algorithm for the single link cluster method. *The Computer Journal*, 16(1):30–34, 1973.
15. Yongqiao Xiao and Margaret H Dunham. Interactive clustering for large databases, July 2000. *Available from http://www.seas.smu.edu/~xiao/publication/oak-tr.ps.*
16. *Oren Zamir and Oren Etzioni. Web document clustering: A feasibility demonstration. In* Proceedings of the 21st International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 46–54, 1998.*
17. *Tian Zhang, Raghu Ramakrishnan, and Miron Livny. Birch: An efficient data clustering method for very large databases. In* Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, pages 103–114, Montreal, Canada, 1996.*

A New Approach For Item Choice Recommendations

Se June Hong, Ramesh Natarajan[1], and Ilana Belitskaya[2]

[1] IBM Thomas J. Watson Research Center
Yorktown Heights, NY 10598
[2] Department of Statistics
Stanford University
Stanford, CA 94325

Abstract. Market basket databases contain historical data on prior customer choices where each customer has selected a subset of items, a market basket, out of a large but finite set. This data can be used to generate a dynamic recommendation of new items to a customer who is in the process of making the item choices. We develop a statistical model to compute the probability of a new item becoming the next choice given the current partial basket, and make a recommendation list based on a ranking of this probability. What makes our approach different from the usual collaborative filtering approaches, is that we account not only for the choice making, or buying, associated with the items present in the partial basket (associative buying), but also for the fact that a customer exercises an independent choice unrelated to the existing partial basket (renewal buying). We compute the probability of both renewal choice and associative choice given the partial basket content, and obtain the probabilities for each item given one of these two buying modes. Our experiments on the publicly available "ms.com" data set shows our new approach yields faster and more accurate prediction compared to other techniques that have been proposed for this problem in the literature.

1 Introduction

Some retail outfits provide carts with displays that provide product information and recommendations as the customer shops. Remote shopping systems allow customers to compose shopping lists through personal digital assistants (PDAs), with interactive recommendations of likely items. Internet commercial sites often provide dynamic product choices as the customer adds items into the virtual shopping cart, or market basket. Internet sites also display dynamically changing set of links to related sites depending on the browsing pattern during a surfing session. All these activities are characterized by the progressive item choices being made by a customer, and the providers' desire to recommend items that are the most likely next choice of the customer.

The past history of the items in each market basket transaction is often available, although most of this data is proprietary. The process of generating recommendations using this data has been called collaborative filtering by Breese et al and Heckerman et al [2,3], and treated in the same way as the modeling of personal preferences of movies or news articles. However, we feel that the market basket recommendation is inherently a different problem. The main reason for this is that preferences are due largely to the taste and interest, whereas the market basket choices are basically a Markov process starting from a null basket state with transitions being made to progressively large–sized baskets. Thus, if there is a sufficient amount of market basket data with information on the order of item choices, one might do a different kind of analysis to arrive at the recommendations in the same spirit as Borges and Levene [4] who develop web user profiling based on an N-gram model. We focus on using the usual market basket data of unordered list of items in each basket, which was a case of "implicit" voting, in contrast to the opinion ratings, the explicit voting, as characterized in [2,3].

Many methods have been proposed for the task of item recommendation. Breese et al [2] summarizes and compares several techniques. In their category of memory based algorithms which includes the first collaborative filtering technique used in GroupLens by Resnick et al [1], each training data baskets are given weight of affinity (various correlation measures) to the partial basket in progress and thus weighted average of all the baskets' choices form the merit for each item that is ranked for recommendations. In their category of model based algorithms, a model is constructed for each item in terms of other items, which is then used to predict the item probabilities given the partial basket as the input pattern. Lawrence et al [5] makes use of the purchase history data of customers. They first form clusters of customers with similar past purchases, and applies association rules from the product domain to make recommendations. Our approach generates a statistical model for the choice behavior in a more straightforward manner.

In this paper, we shall assume that the market basket data for training is in the form of 0/1 matrix $\mathbf{M}$ even though the actual data may be in some sparse matrix representation. Each row is a basket and each column is an item. The value of $\mathbf{M_{ij}}$ is 1 if and only if basket i contains item j. There are total of $\mathbf{n}$ baskets and $\mathbf{m}$ items. We also assume that no basket is empty, and each item is contained in some basket.

In the following section, we present the rationale for the separation of renewal and associative choices. In Section 3, we develop the statistical model for these modes of buying. Section 4 describes how the model parameters are estimated from the market basket data. A summary of our algoritm is presented in Section 5. We report on our experiment on the ms.com data in Section 6, followed by a conclusion section.

2 Renewal vs. associative item choices

When a customer embarks on making some number of choices, be it purchasing or visiting web sites, we argue that not all items in the basket are selected because of their association with some other item already in the basket. Starting from a null basket, some need drives the customer to select the first item, which is due to some independent need that we call the **renewal choice**. After the first item, a customer may stop, or select another item by association or by another renewal choice, iteratively.

Let $N(k)$ be the number of baskets in the basket data that has exactly k items and $N(k^+)$ be the number of baskets with k or more items. We have observed that for most basket data, the "reset rate", $R(k) = N(k)/N(k^+)$ is almost constant or a slowly decreasing function of k. In fact, for the only publicly available data, ms.com, in UC Irvine repository, the $R(k)$ for $1 \leq k \leq 10$ in the test set of 5000 baskets are 0.31, 0.35, 0.33, 0.34, 0.33, 0.32, 0.34, 0.29, 0.29 and 0.29. The weighted average of these ratios using $N(k^+)$ as weight yields 0.3267. For the training data of 37211 baskets, similar weighted average is 0.3337 and each $R(k)$ is even closer to this constant. We see that when a customer has a basket of size k, $R(k)$ fraction of the time the basket was terminated, and $1 - R(k)$ fraction of the time further choices were made. In the case of ms.com data, a fixed $R = 1/3$ can be used for the basket termination probability, and the remaining 2/3 as continuation. In such constant renewal process with a constant reset rate R, the proportion of size k baskets would be $R(1 - R)^{k-1}$ and the mean basket size would be

$$k = \sum_{k=1}^{\infty} kR(1 - R)^{k-1}$$

For the ms.com data with $R = 1/3$, $k = 3$ which is very close to the actual mean basket sizes: 3.038 for the test set and 3.016 for the training set.

It is clear that when a basket begins from scratch there cannot be any associated choice and whenever a basket would terminate, the next choice would be as if the basket is just beginning. This is the renewal buying choice. In the sequel, all the probabilities we develop will be for buying (or choice making) with implicit given condition that there will be a next choice. Also we shall use the simple case of constant reset rate to present our method, although it can easily accommodate separate $R(k)$ for each partial basket size k.

Now, let n_j be the number of baskets where item j is present and n'_j be the number of baskets where item j is the only item. Every time a singleton choice is made, it is a sure case of renewal buying, so we can lower bound the probability of renewal, i.e., the proportion of singleton choices over all choices made, as

$$P(renew) \geq \frac{\sum_j n'_j}{\sum_j n_j}.$$

In the case of ms.com data, this value is about 0.1.

3 Statistical modeling for the next choice probability

We start out with a current basket **B** which contains b items, $i_1, i_2, ..., i_b$, where b may be 0 if the basket is just beginning. Such case will be called a null basket, and it will be dealt with separately. We denote by $P(j|\mathbf{B})$ the probability that the next choice will be item j given the current partial basket. For the two modes of making a choice (or buying), we have

$$P(j|\mathbf{B}) = P(j, asso|\mathbf{B}) + P(j, renew|\mathbf{B}).$$

Using Bayes rules, we re-express this as

$$P(j|\mathbf{B}) = P(j|asso, \mathbf{B})P(asso|\mathbf{B}) + P(j|renew, \mathbf{B})P(renew|\mathbf{B}), \qquad (1)$$

for all $j \notin \mathbf{B}$, or $j \in \mathbf{B}$, the complement set. And since choice is made either by renewal or by association,

$$P(asso) + P(renew) = 1, \quad \text{and also} \quad P(asso|\mathbf{B}) = 1 - P(renew|\mathbf{B}). \qquad (2)$$

And in the case of renewal buy, the basket content is immaterial and, hence,

$$P(j|renew, \mathbf{B}) = P(j|renew) = \frac{P(renew|j)P(j)}{P(renew)}, \qquad (3)$$

where $P(j)$ is the probability of item j being bought.

Now we make a simple but reasonable assumption about the choice behavior that we call **single item influence assumption**. That is for both renewal or associative buying, the next choice is the result of such tendencies due to each of the items in the current basket considered singly. In other words, an associative choice would be due to an association to some single item in the basket and not because of multiple items taken together. Likewise, each item exerts its own tendency for renewal, even though when the basket grows large, the customer's financial or time resource may limit the next action. We make further simplifying assumptions for the way each item's influence is aggregated. These assumptions allow for an efficient computation without apparently affecting the accuracy of our approach.

For renewal, we assume that the least renewal tendency among all the basket items dictates the final renewal:

$$P(renew|\mathbf{B}) = \min_{1 \le k \le b} P(renew|i_k), \qquad (4)$$

which will be obtained in a manner described later. For association, we assume that maximum preference to associatively select an item j among each item in the basket determines the overall preference for $j \notin \mathbf{B}$. That is, in pre-normalized form,

$$P'(j|asso, \mathbf{B}) = \max_{1 \le k \le b} P(j|asso, i_k). \qquad (5)$$

This quantity is 0 for all the items that are in the partial basket in progress. Normalizing for probability, we have

$$P(j|asso, \mathbf{B}) = \frac{P'(j|asso, \mathbf{B})}{\sum_k P'(k|asso, \mathbf{B})}. \qquad (6)$$

Again using Bayes rule, we rewrite the quantity $P(j|asso,i)$ of equation (5) (using i for i_k) as

$$P(j|asso,i) = \frac{P(j|i)P(asso|j,i)}{P(asso|i)} = \frac{P(j|i)(1-P(renew|j,i))}{(1-P(renew|i))}.. \tag{7}$$

Here we are make use of (i,j) pair statistics that are reasonably available from the data. If there is an abundance of triplet occurances in the data, one might expand the analysis to include triplet statistics or even higher order statistics, by revising the single item influence assumption to appropriately include more items. This would increase the computation.

When the basket is just beginning from a null basket, the customer is precisely at the renewal point. Therefore, for the null basket, i.e. $\mathbf{B} = \emptyset$, equation (1) is specialized by use of equation (3):

$$P(j|null) = P(j|renew). \tag{8}$$

4 Estimation of model parameters from data

The model parameters we have developed in the previous section are estimated from the training data $\mathbf{M}$ and stored for use for future recommendation phase. (In gathering the statistics, we make use of the row sums of $\mathbf{M}$ to facilitate various computations). The probability of item j being chosen is just the raw item popularity measure:

$$P(j) = \frac{n_j}{\sum_k n_k}, \tag{9}$$

or, optionally, with a Laplace correction for small statistics as

$$P(j) = \frac{n_j + 1}{\sum_k (n_k + 1)},$$

although this correction did not make any noticeable difference for the ms.com data. For the renewal probability, we use the lower bound discussed earlier in the same spirit as the minimal renewal aggregation:

$$P(renew) = \frac{\sum_j n'_j}{\sum_j n_j}. \tag{10}$$

For $P(renew|i)$, the estimation is done in two stages. Firstly, among the baskets that contained item i, n'_i cases are certainly renewal. Secondly, among the remaining $n_i - n'_i$ cases, we can estimate that on the average a $P(renew)$ proportion would be renewal, hence

$$P(renew|i) = \frac{n'_i}{n_i} + (1 - \frac{n'_i}{n_i})P(renew). \tag{11}$$

$P(j|renew)$ of equation (3) can now be precomputed from these estimates. (Once the basket $\mathbf{B}$ is given, this quantity will be modified as described later).

To estimate $P(j|i)$ in equation (7), we make use of a conceptual submatrix $\mathbf{M}_i$ of $\mathbf{M}$, obtained by selecting only the rows that has item i. Similarly, we denote by n_{ji} the j'th column sum of $\mathbf{M}_i$, i.e., the number of times j was chosen along with i. We set $P(i|i)$ to 0 and pre-compute for all $(i, j \neq i)$

$$P(j|i) = \frac{n_{ji}}{\sum_k n_{ki}}. \tag{12}$$

Of course, many of these quantities are 0 when $n_{ji} = 0$, so the computation can skip over them and the result can be stored in a sparse matrix form. Again, similar to before, we denote by n'_{ji} the number of rows whose row sum in $\mathbf{M}_i$ is exactly 2, i.e., no other item was in the basket. Following a similar line of reasoning as in $P(renew|i)$ estimation, we have

$$P(renew|j,i) = \frac{n'_{ji}}{n_{ji}} + (1 - \frac{n'_{ji}}{n_{ji}})P(renew). \tag{13}$$

These can be evaluated as needed, from just the n'_{ji}/n_{ji} values stored in a sparse representation. We observed that only 2.6% of the n'_{ji} values and 28% of the n_{ji} values are non-zero for the ms.com data set, and for data sets with even more items we expect the sparsity to be even greater. Now, with these estimates, $P(j|asso, i)$ of equation (7) can be computed and stored.

When a recommendation is to be made for a basket $\mathbf{B}$ in progress, we first compute $P(j|asso, \mathbf{B})$ of equation (7) via $P'(j|asso, \mathbf{B})$ of equation (5) using the precomputed components above, and set its values to 0 for all $j \in \mathbf{B}$ for these items are not recommended again. Now we can normalize P' values to

$$P(j|asso, \mathbf{B}) = \frac{P'(j|asso, \mathbf{B})}{\sum_k P'(k|asso, \mathbf{B})}. \tag{14}$$

Likewise, since no item already in the basket is to be recommended, we set the values of $P(j|renew)$ to 0 for all $j \in \mathbf{B}$, and normalize these modified renewal buying probabilities to sum to 1, before finally computing the total probability $P(j|\mathbf{B})$ for each $j \notin \mathbf{B}$ by equation (1). The $P(j|\mathbf{B})$ values are ranked in descending order, and the top several of this ranked list are "recommended". Alternatively, if there are some other merit values associated with each item, related to profit potential or inventory reduction, these can be turned into the expected merit value by combining with the item probability, and ranking the items according to these expected merits for recommendation.

5 Summary of the new recommendation algorithm

Our algorithm is comprised of three steps. The first two steps use the market basket information in the training data base to build our statistical model described in section 4. In the first step, basic statistics are collected, which are then used in the second step to pre-compute the model parameters discussed above. (Use of row sums is implicit here). The third step uses the model and the current basket information to produce a preference ranking for the items not in the basket. Using the notations introduced earlier, we now summarize the algorithm.

1. Collect statistics from training data
   (a) For each item j, obtain n_j the number of baskets with item j purchased.
   (b) For each item j, obtain n'_j the number of baskets with item j being the sole item purchased.
   (c) For each pair of items i and j, obtain the number of market baskets n_{ji} with items j and i purchased together.
   (d) For each pair of items i and j, obtain the number of market baskets n'_{ji} with items i and j being the only two items purchased.
2. Pre-compute model parameters
   (a) Compute $P(renew) = \sum_k n'_k / \sum_k n_k$ (equation (10)).
   (b) For each item j, compute $P(j) = n_j / \sum_j n_j$ (equation (9)).
   (c) For each item j, compute $P(renew|j) = \frac{n'_j}{n_j} + P(renew)(1 - \frac{n'_j}{n_j})$ (equation (11)).
   (d) For each item j, compute $P'(j|renew) = P(renew|j)P(j)/P(renew)$ (equation (3)).
   (e) For each pair of items i and j with $n_{ij} \neq 0$, compute $P(j|i) = n_{ji} / \sum_k n_{ki}$ (equation (12)).
   (f) For each pair of items i and j with $n_{ij} \neq 0$, compute $P(renew|j,i) = \frac{n'_{ji}}{n_{ji}} + P(renew)(1 - \frac{n'_{ji}}{n_{ji}})$ (equation (13))
   (g) For each pair of items i and j with $n_{ij} \neq 0$, compute $P(j|asso,i) = P(j|i) \times (1 - P(renew|j,i))/(1 - P(renew|i))$ (equation (7)).
3. Recommendation ordering for a given partial basket

   Given a partial basket $\mathbf{B}$ let $\mathbf{B}$ be the complementary set of items not in $\mathbf{B}$
   (a) If $\mathbf{B}$ is empty, then sort items in order of decreasing $P'(j|renew)$ and return this as the item preference ordering.
   (b) If $\mathbf{B}$ is non-empty, then
      i. Compute $P(renew|\mathbf{B}) = \min_{j \in \mathbf{B}} P(renew|j)$ (equation (4))
      ii. Compute the normalization factor $\sum_{k \in \mathbf{B}} P'(k|renew)$.
      iii. For each item $j \in \mathbf{B}$, compute $P(j|renew) = P'(j|renew) / \sum_{k \in \mathbf{B}} P'(k|renew)$.
      iv. For each item $j \in \mathbf{B}$, compute $P'(j|asso,\mathbf{B}) = \max_{i \in \mathbf{B}} P(j|asso,i)$ (equation (5)).
      v. Compute the normalization factor $\sum_{j \in \mathbf{B}} P'(j|asso,\mathbf{B})$.
      vi. For each item $j \in \mathbf{B}$, compute $P(j|asso,\mathbf{B}) = P'(j|asso,\mathbf{B}) / \sum_{j \in \mathbf{B}} P'(j|asso,\mathbf{B})$ (equation (6)).
      vii. For each item $j \in \mathbf{B}$, compute $P(j|\mathbf{B}) = P(j|asso,\mathbf{B})(1 - P(renew|\mathbf{B})) + P(j|renew)P(renew|\mathbf{B})$ (equation (1)).

viii. Sort items in order of decreasing $P(j|\mathbf{B})$ and return this as the item preference ordering.

6 Experiments and Performance evaluation

Although three example cases were discussed in [2,3], only the ms.com data is publicly available. Also, due to the proprietary nature of market basket data, even the ones we have access to are confidential. However, the conclusions drawn from the ms.com data seem to be generally applicable, and the training/test set breakdown of the ms.com data seems to be typical. We have ran the recommendation on this data 100 times because of relatively large variance observed in the so called cfaccuracy measure [2,3] among the runs. The standard deviation of these runs on our method is shown in Table 1). This is due to the random partitioning of the test baskets into the partial basket in progress and the remaining items to be "predicted". We have also combined the training and test sets and randomly repartitioned it into the same size training and test set breakdown five times, and on each pair ran our recommendation 50 times. The variation among these five break downs are very small and we report only the average of all of them. We followed the same experimental regimen as in [2,3] and used the most sensible measure, the cfaccuracy, which we rephrase here.

When a partial basket is used to rank the remaining items, the best possible outcome would be that the remaining items in the test basket would be assigned the top so many ranks. A ranked item with rank r is credited $2^{-r/5}$ and the cfaccuracy of a recommendation on t remaining items $(i_1, i_2, ..., i_t)$ in the basket with ranks, $r(i_j)$ is given as a percentage of the total credit of the recommendation to the best possible, as

$$cfaccuracy = 100 \frac{\sum_{j=1}^{t} 2^{-r(i_j)/5}}{\sum_{j=0}^{k-1} 2^{-j/5}}$$

If only one item is to be "predicted", a rank of 0 (the best) would yield 100%, and a rank of 5 (the 6'th best) would yield 50%. This gives some idea as to how many of the top items should be actually presented to the customer to get a hit.

For the "Givenj" testing, the subset of the 5000 test baskets with the basket size greater than j are processed. For each basket in this subset, j items are randomly selected as in the partial basket and the cfaccuracy for the remaining items in the basket is computed. Then the cfaccuracies are averaged to give one fugure for the "Givenj". Table (1) gives the average cfaccuracy of our 100 runs on the original split, along with the single run results reported in [3] for comparison. The two models, BayesNet and DependencyNet, are said to be the best of their experience. Also given are the 5×50 run results of our method. The results on these re-partitioned training/test set are not much different from the original split.

More recently, another model-based approach of using a linear item presence model was reported by Iyengar and Zhang [6]. These results are also shown in Table (1).

Table 1. Cfaccuracy of recommendations for ms.com data

	Given 0	Given 2	Given 5	Given 10	AllBut1		
No. of cases	5000	2213	657	102	3453	Avg	
$P(j)$ base	52.18	45.52	38.57	31.49	45.40		"Original training/test set,
$P(j\|renew)$	**52.28**	46.65	42.85	39.16	47.82		100 runs average"
$P(j\|\mathbf{B})$	**52.28**	**59.51**	**57.52**	53.83	62.81	**58.88**	
std. dev.	0.0	0.46	0.81	2.24	0.48		
$P(j)$ base	52.12	45.58	39.32	32.38	45.41		"Random training/test set,
$P(j\|renew)$	52.10	46.61	43.83	40.57	47.84		5x50 runs average"
$P(j\|\mathbf{B})$	52.10	59.31	57.85	54.10	62.58		
BayesNet	—	53.18	52.48	51.64	66.54	52.97	"Results of [3] on the original
DepNet	—	52.68	52.54	51.48	**66.60**	52.61	training/test set, one run"
Linear	—	55.70	57.50	**57.00**	64.10	56.14	Least Squares (linear) from [6]
Baseline	52.18?	43.37	39.34	39.32	49.77	—	$P(j)$ only base from [3]

We see from the table that our method is generally better (bold numbers) except in the AllBut1 category. There are two reasons for this. First, the model based method becomes more accurate as more inputs are known, but it happens only when the basket sizes are larger than the average. Second, there is usually a few % cfaccracy fluctuation in any given run due to the way the test basket is split for partial basket and the target items. That aside, we can form a weighted average of the cfaccuracy figures for various methods. We do this just for the 3 cases: Given2, Given5 and Given10. (The AllBut1 case is actually overlapping these three cases, and Given0 was not reported in [2,3,6]). These averages are shown in the last column (avg) of the table. The obvious advantage of our approach will prevail even if we had averaged all the Givenj recommendations including the three shown here.

Our method takes about 3 minutes to build the model, and 2 minutes to process the test set for one run for all 5 Given categories. Not counting the Given0 cases, there are 6425 recommendations per run. Our program is in APL running on a 63 MHz IBM RS/6000 machine. This is much faster than the run times for BayesNet and DependencyNet models running on a 300 MHz Pentium processor, where it is reported [3] that about 2 minutes was required to build the model, and about 10 recommendations could be processed per second

7 Conclusion

The new approach we propose is more efficient and more accurate than other state of the art methods, especially when the partial basket size is small (which is usually the case of importance in practise). We conclude that the advantage is due to the proper recognition of the choice process as a renewal process and modeling the underlying statistical mechanism.

Much more work is to be done to work out the details of using sparse representation of the model and in adopting the approach to a very large item set situation (ms.com has less than 300 items) so that it will scale better. The run time and storage requirement of our method, as presented here, are both of $O(\mathbf{m}^2)$. For very large number of items, many computational tricks as well as some modification of the algorithm will be necessary. Some of the modifications we foresee are, clustering the items to some manageable number of super-items, implementing sparse techniques, and post refinment of the recommended super-items into individual items.

References

1. Resnick P., Iacocou N., Suchak M, Berstrom P. and Riedl J.,"Grouplens: An Open Architecture for Collaborative Filtering of Netnews," Proc. of the ACM 1994 Conf. On Computer Supported Cooperative Work, pp.175-186, ACM, New York, 1994.
2. Breece J., Heckerman D. And Kadie C.,"Empirical Analysis of Predictive Algorithms for Collaborative Filtering," Proc. of Fourteenth Conf. on Uncertainty in Artificial Intelligence, Morgan Kaufman, Madison Wisc., 1998.
3. Heckerman D., Chickering D.M., Meek C., Rounthwaite R. And Kadie C,"Dependency Network for Collaborative Filtering and Visualization", Proc. of Sixteenth Conf. on Uncertainty in Artificial Intelligence, Stanford, CA, pp. 264-273, 2000
4. Borges J. and Levene M.,"Data Mining of User Navigation Patterns," Web Usage Analysis and User Profiling", WEBKDD-99 Workshop revised papers, Lecture Notes in Artificial Intelligence, Vol. 1836, pp. 92-110, 2000.
5. Lawrence R.D., Almasi G.S., and Kotlyar V. and Duri, S. S.,"Personalization of Supermarket Product Recommendations", Journal of Data Mining and Knowledge Discovery, Vol. 5, Num. 1/2, pp. 11-32, Jan/Apr 2001.
6. Iyengar V.S., Zhang T.,"Emprical Study of Recommender Systems using Linear Classifiers", to appear in Proc. of PAKDD-01, April 2001.

RecTree: An Efficient Collaborative Filtering Method

Sonny Han Seng Chee, Jiawei Han, and Ke Wang

Simon Fraser University, School of Computing Science, Burnaby, B.C., Canada
{schee,han,wangk}@cs.sfu.ca

Abstract. Many people rely on the recommendations of trusted friends to find restaurants or movies, which match their tastes. But, what if your friends have not sampled the item of interest? Collaborative filtering (CF) seeks to increase the effectiveness of this process by automating the derivation of a recommendation, often from a clique of advisors that we have no prior personal relationship with. CF is a promising tool for dealing with the information overload that we face in the networked world.

Prior works in CF have dealt with improving the accuracy of the predictions. However, it is still challenging to scale these methods to large databases. In this study, we develop an efficient collaborative filtering method, called *RecTree* (which stands for RECommendation Tree) that addresses the scalability problem with a divide-and-conquer approach. The method first performs an efficient k-means-like clustering to group data and creates neighborhood of similar users, and then performs subsequent clustering based on smaller, partitioned databases. Since the progressive partitioning reduces the search space dramatically, the search for an advisory clique will be faster than scanning the entire database of users. In addition, the partitions contain users that are more similar to each other than those in other partitions. This characteristic allows RecTree to avoid the dilution of opinions from good advisors by a multitude of poor advisors and thus yielding a higher overall accuracy.

Based on our experiments and performance study, *RecTree* outperforms the well-known collaborative filter, *CorrCF*, in both execution time and accuracy. In particular, *RecTree*'s execution time scales by $O(n\log_2(n))$ with the dataset size while *CorrCF* scales quadratically.

1 Introduction

In our daily life, virtually all of us have asked a trusted friend to recommend a movie or a restaurant. The underlying assumption is that our friend shares our taste, and if she recommends an item, we are likely to enjoy it. If a friend consistently provides good recommendations, she becomes more trusted, but if she provides poor recommendations, she becomes less trusted and eventually ceases to be an advisor. Collaborative filtering (CF) describes a variety of processes that automate the interactions of human advisors; a collaborative filter recommends items based upon the opinions of a clique of human advisors. Amazon.com and CDNow.com are two well known e-commerce sites that use collaborative filtering to provide recommendations on books, music and movie titles; this service is provided as a means to promote customer retention, loyalty and sales, etc. [13].

Example 1. A ratings database records a patron's reaction after viewing a video. Users collaborate to predict movie preference by computing the average rating for a movie from among their friends. A subset of the database is shown below where Sam and Baz have indicated a common set of friends. The average rating of *Matrix* is 3 while *Titanic* is 14/4. Therefore, *Titanic* would be recommended over *Matrix* to Sam and Baz.

Table 1: Higher scores indicate a higher level of enjoyment in this ratings database.

		Titles				
		Speed (A)	Amour (R)	MI-2 (A)	Matrix (A)	Titanic (R)
Users	Sam	3	4	3		
	Bea	3	4	3	1	1
	Dan	3	4	3	3	4
	Mat	4	2	3	4	3
	Gar	4	3	4	4	4
	Baz	5	1	5		

This simplistic approach falls well short of automating the human advisory circle. In particular, the group average algorithm implicitly assumes that all advisors are equally trusted and consequently, their recommendations equally weighted. An advisor's past performance is not taken into account when making recommendations. However, we know that in off-line relationships, past performance is extremely relevant when judging the reliability of recommendations. Equally problematic is that the group average algorithm will make the same recommendation to all users. Baz, who has very different viewing tastes from Sam, as evidenced by his preference for action over romantic movies (as indicated by the letter A and R following each of the titles) will nevertheless be recommended *Titanic* over *Matrix*. Collaborative filters aim to overcome these shortcomings to provide recommendations that are personalized to each user and that can adapt to a user's changing tastes. □

Memory-based algorithms [3] are a large class of collaborative filters that take a list of item endorsements or a ratings history, as input for computation. These algorithms identify advisors from similarities between rating histories and then generate a recommendation on an as-yet unseen item by aggregating the advisors' rating. Memory-based collaborative filters differ in the manner that ratings are defined, the metric used to gauge similarity, and the weighting scheme to aggregate advisors' rating.

In the well-known correlation-based collaborative filter [11], that we call *CorrCF* for brevity, a 5-point ascending rating scale is used to record user reactions after reading Usenet items. Pair-wise similarity, $w_{u,a}$, between the user, u, and his potential advisor, a, is computed from Pearson correlation of their rating histories.

$$w_{u,a} = \sum_{i \in Y_{u,a}} \frac{(r_{u,i} - \bar{r}_u)(r_{a,i} - \bar{r}_a)}{\sigma_u \sigma_a \, | Y_{u,a} |} \tag{1}$$

where $r_{u,i}$ and $r_{a,i}$ is the user and advisor rating for item i while $\bar{r}_u$ and $\bar{r}_a$ is the mean ratings of each user; σ_u and σ_a is the standard deviation of each user's rating history, and $Y_{u,a}$ is the set of items that both the user and his advisor have rated. A recommendation, $p_{u,j}$ is then generated by taking a weighted deviation from each advisor's mean rating.

$$p_{u,j} = \bar{r}_u + \tfrac{1}{\alpha} \sum_{i \in Y_{u,a}} (r_{a,i} - \bar{r}_a) \cdot w_{u,a} \qquad (2)$$

where α is a normalizing constant such that the absolute values of the correlation coefficients, and $w_{u,a}$ sum to 1.

The computation of the similarity coefficients can be viewed as an operation to fill in the entries of an *n* by *n* matrix where each cell stores the similarity coefficient between each user and his *n-1* potential advisors. Each row of the matrix requires a minimum of one database scan to compute and to fill the entire matrix of *n* rows therefore requires $O(n^2)$ operations. The computation of these similarity coefficients is the performance bottleneck in all previously published memory-based algorithms.

RecTree solves the scalability problem by using a divide-and-conquer approach. It dynamically creates a hierarchy of cliques of users who are approximately similar in their preferences. *RecTree* seeks advisors only from within the clique that the user belongs to and since the cliques are significantly smaller than the entire user database, *RecTree* scales better than other memory-based algorithms. In particular, creating more cliques as the dataset size increases allows *RecTree* to scale with the number of cliques rather than the number of users.

In addition, the partitions contain users that are more similar to each other than to users of other partitions. This characteristic allows *RecTree* to avoid the dilution of opinions from good advisors by a multitude of poor advisors – yielding a higher overall accuracy. The trick then is to create cohesive cliques in an economical manner.

This paper is organized as follows. Section 2 reviews related work. Section 3 details the *RecTree* algorithm. Section 4 describes the implementation of the *RecTree* algorithm and the experimental methodology. Section 5 compares *RecTree*'s performance against *CorrCF*. We conclude in Section 6 with a discussion of the strengths and weaknesses of our approach and the direction of future research.

2 Related Work

Two of the first automated collaborative filtering systems use Pearson correlation to identify similarities between users of Usenet[11] and music album aficionados[14]. In [14], the constrained Pearson correlation is introduced to account for the implicit positivity and negativity of a rating scale. Ringo also provides an innovative solution that inverts the basic CF approach; music albums are treated as 'participants' that can recommend users to other music album participants.

When the rating density is low, most CF systems have difficulty generating accurate recommendations [11] [5]. Unlike the problem of scalability, however, rating sparsity is an open issue that has received significant research attention. [12] and [5] attempt to ameliorate this issue by using bots and agents to artificially increase the rating density. Bots assign ratings based on criteria such as the number of spelling errors, the length of the Usenet message, the existence of included messages [12] or the genre of the movie title [5]. Agents are trained, using IF techniques, to mimic the rating distribution of each user. An agent regenerates its ratings as it becomes better trained which may force large portions of the similarity matrix to be updated [5]. In both of these works, the relevancy of the bots' and agents' ratings to a particular user is decided by the CF system as it identifies potential advisors.

In their recent paper, Goldberg et al. [4] describe Eigentaste, which for certain domains does not suffer from the sparsity and scalability problems. They note that rating sparsity is introduced during the profiling stage when users are given the freedom to select the items they rate. In contrast, the Eigentaste algorithm forces participants to rate all items in a *gauge set*. The dimensionality of the resulting dense rating matrix is reduced using principal component analysis to the first two dimensions. All of the users are then projected onto this eigen-plane and a divisive clustering algorithm is applied to partition the users into neighbourhoods. When a new user joins the system their neighbourhood is located by projecting their responses to the gauge set onto the eigen-plane. A recommendation is generated by taking neighbourhood's average rating for an item. Eigentaste is a linear collaborative filter and requires $O(j^2n)$ time to compute its cluster structure. For small values of j, the size of the gauge set, Eigentaste can be very fast.

Eigentaste is however limited in that it requires the definition of a gauge set. In the Jester recommendation service, the gauge set consists of a set of jokes. After reading a joke, each user can immediately supply a rating. However, there are few domains where the items of interest can be consumed so quickly and evaluated.

It is worth noting that many e-commerce sites provide a simplified form of collaborative filtering that is based on the complementary technologies of data warehousing and on-line analytical processing (OLAP). Often-seen examples of OLAP style collaborative filtering are the factoids that attempt to cross-sell/up-sell products: *Item X has been downloaded Z times*. These rudimentary filters make the implicit assumption that all users are equally good advisors to the active user. A more sophisticated approach would be to mine patterns from the database and data warehouse [7] and to use these as the basis of a recommendation to the user.

3 The *RecTree* Algorithm

RecTree is the acronym for a new data structure and collaborative filtering algorithm called the RECommendation Tree. The *RecTree* algorithm partitions the data into cliques of approximately similar users by recursively splitting the dataset into child clusters. Splits are chosen such that the intra-partition similarity between users is maximized while the inter-partition similarity is minimized. This yields relatively small cohesive neighbourhoods that *RecTree* uses to restrict its search for advisors – which represent the bottleneck in memory-based algorithms. *RecTree* achieves its $O(n\log_2(n))$ scale-up by creating more partitions to accommodate larger datasets - essentially scaling by the number of partitions rather than the number of users.

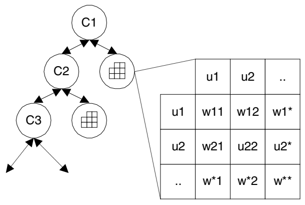


Fig. 1: The *RecTree* data structure.

Prediction accuracy deteriorates when a large number of lowly correlated users contribute to a prediction. Herlocker et al. [8] suggest that a multitude of poor advisors can dilute the influence of good advisors on computed recommendations.

The high intra-partition similarity between users makes *RecTree* less susceptible to this dilution effect – yielding a higher overall accuracy.

The chain of intermediate clusters leading from the initial dataset to the final partitioning is maintained in the *RecTree* data structure, which resembles a binary tree. Within each leaf node, computing a similarity matrix between all members of that clique identifies advisors. *RecTree* then generates predictions by taking a weighted deviation from each clique's advisor ratings using (2).

Lemma 1. The training time for a partitioned memory-based collaborative filter is O(nb), where n is the dataset size if the partitions are approximately b in size and there are n/b partitions.
Proof. Training consists of computing a similarity matrix for each of the n/b partitions. Each partition is approximately b is size and computing the similarity matrix requires O(b^2) to compute. Computing all n/b similarity matrices is therefore O(nb). □

By fixing the partition size, Lemma 1 indicates that the training time for a memory-based collaborative filter can be linear in n.

3.1 Growing the *RecTree*

The *RecTree* is grown by recursively splitting the data set until a good partitioning of the data is obtained. It seems obvious that a clustering algorithm would be the ideal candidate for creating these partitions. Indeed, recent results have extended the applicability of clustering algorithms to high dimensions and very large disk-resident data sets [6] [2] [1]. However, given the sparsity of rating data and the low cost of RAM it is quite feasible to load all of the rating data into memory[1]; a fast in-memory clustering algorithm, such as KMeans [9] would therefore be appropriate for our needs.

KMeans begins its clustering by selecting k initial seeds as the temporary cluster centers and then assigning users to the cluster that they are closest to. The centroid of each cluster is then taken as the new temporary center and users are reassigned. These steps are repeated until the change in centroid positions fall below a threshold.

KMeans has a time complexity of O(k^2n) where k is the number of clusters and n is the dataset size. A naïve application of KMeans to create a proportionate number of cliques to match an increase in dataset size would yield cubic scale-up. Rather we employ KMeans as a procedure in a hierarchical clustering algorithm, which recursively splits the dataset into two child clusters as it constructs the *RecTree* from the root to its leaves. Since k is always 2 in this instance, KMeans is guaranteed to execute in time linear with dataset size. The procedure for constructing the *RecTree* is outlined in Algorithm 1.

Our purpose in selecting a clustering algorithm is neither to locate nor to identify the cluster structure in the dataset. We partition the data because we want to improve

[1] The EachMovie service over the course of 18 months accumulated over 70,000 users and an inventory of over 16000 movies, yet its entire rating database can be compressed and loaded into only 6 megabytes of RAM.

the execution time of the collaborative filter. The conditions for Lemma 1 guarantee linear training time but also restrict a clustering algorithm from obtaining optimal clusters. Despite this, our performance analyses show that *RecTree* is more accurate than un-partitioned collaborative filtering via *CorrCF*.

The *ConstructRecTree()* procedure is called recursively for each child cluster, *childClusterDataSet*, that is created by *KMeans()*. The tree continues to grow along a branch until the cluster is smaller than *partitionMaxSize*, or the branch depth, *curDepth*, exceeds the *maxDepth* threshold. The first condition ensures that all the cliques are approximately equal in size, which is essential to our efficient collaborative filter. The second condition ensures that *ConstructRecTree()* does not pursue a pathological partitioning.

```
Algorithm 1. ConstructRecTree(parent, dataSet,
partitionMaxSize, curDepth, maxDepth)
Input: parent is the parent node from which the
dataSet originates.  partitionMaxSize is the maximum
partition size and curDepth is the depth of the
current branch.  maxDepth is the maximum tree depth.
Output: The RecTree.
Method:
1. Create a node and link it to parent.
2. Assign dataSet to node.
3. If  SizeOf(dataSet)  ≤ partitionMaxSize  OR
   curDepth > maxDepth then
      ComputeCorrelationMatrix(dataSet); RETURN.
4. curDepth++.
5. Call KMeans (dataSet, numberofClusters=2)
6. For each child cluster resulting from KMeans:
      Call ConstructRecTree(node,
childClusterDataSet, partitionMaxSize, curDepth,
maxDepth).
```

Lemma 2. The *RecTree* data structure is constructed in $O(gn\log_2(n/b))$, if the maximum depth is $g\log_2(n)$. n is the dataset size, b is the maximum partition size and g is a constant.

Proof. We established in Lemma 1 that *RecTree*'s training phase (step 4) is linear. All that remains is to show the complexity of creating the partitions. At level one, the cost of creating two partitions is qn, where q is a constant and n is the dataset size. At each subsequent level of the tree, the complexity of building the branches is $qn_1 + qn_2 + .. qn_t$, where t is the number of partitions on a level. Since $n = n_1 + n_2 + \ldots n_t$, the cost of each subsequent level is also qn. For a balanced tree the maximum depth is $\log_2(n/b)$, which yields a complexity of $O(n\log_2(n/b))$. For an unbalanced tree, the maximum depth is n/b, which yields a complexity of $O(n^2/b)$ at worst. Since we constrain the maximum depth at $g\log_2(n/b)$, the total complexity is at worst $O(gn\log_2(n/b))$. □

3.2 Default Voting

In the collaborative filters proposed by [11] and [14], pair-wise similarity is computed only from items that both users have rated. If the item intersection set is small, then the coefficients are poor measures of similarity as they are based on few comparisons. Furthermore, an emphasis on intersection set similarity neglects the global rating behaviour that is reflected in a user's entire rating history. [8] accounts for small intersection sets by reducing the weighting of advisors who have fewer than 50 items in common. We approach this issue by extending each user's rating history with the clique's averages. This default voting effectively increases the similarity measure to cover the union of ratings rather than just the intersection set.

3.3 Generating a Recommendation

RecTree generates a prediction for the active user by locating his clique and then applying the weighted deviation from mean method as shown in (2).

Example 2. Table 2a shows the correlation between Sam and his friends. Without default voting, Bea and Dan are indistinguishable; they have voted identically on the three movies that Sam has seen. However, the standard deviation and average rating indicate that Bea is more volatile in voting than Dan or Sam. With default voting, the correlation is computed over the entire voting histories and captures the differences in global behaviour; Dan is assigned a higher similarity coefficient than Bea.

The ConstructRecTree algorithm partitions the users into the two groups consisting of (Sam, Bea, Dan) and (Mat, Gar, Baz). Table 2b shows the movie predictions that *RecTree* generates using default voting and this partitioning. *RecTree*'s predictions are in-line with our expectations and recommend the romantic title to Sam and the action title to Baz. In comparison to *CorrCF*, *RecTree* more clearly capture Sam's tastes by predicting a larger difference in preference for *Titanic* to *Matrix*. *RecTree* generates each of these predictions from considering only 2 advisors, while *CorrCF* considered 4 advisors. The reduced search space results in better execution time for *RecTree* and since the intra-partition similarity is high, the dilution of good advisors by a multitude of poor advisors is avoided.

		Titles							
		Speed	Amour	MI-2	Matrix	Titanic	Std.		Corr to
		(A)	(R)	(A)	(A)	(R)	Dev	Avg	Sam
Users	Sam	3	4	3	3	3.75	0.5	3.4	1
	Bea	3	4	3	1	1	1.3	2.4	0.21
	Dan	3	4	3	3	4	0.5	3.4	0.98
	Mat	4	2	3	4	3	0.8	3.2	-0.83
	Gar	4	3	4	4	4	0.4	3.8	-0.75
	Baz	5	1	5	3	3.75	1.7	3.6	-0.74

(2a)

		Titles	
		Matrix	Titanic
		(A)	®
	Sam	2.72	3.55
	Baz	4.15	3.73

(2b)

Fig. 2a. Similarity coefficients computed with default voting capture global rating behavior. **2b)** RecTree's movie recommendations.

4 Experiments and Performance Analysis

We base our performance study on a comparison with the well-known *CorrCF* algorithm. This filter has been demonstrated to be the most accurate of the memory-based filters [3] and has been incorporated into the GroupLens [10] and MovieLens recommendation systems [12].

The *maxDepth* parameter is constrained by 0 to be $gn\log_2(n/b)$. For these experiments, we found that setting g to a value of two protected the *RecTree* algorithm from pathological data distributions while creating partitions efficiently. The performance analysis shows that this choice not only yields better than quadratic scale-up, but also improves in accuracy over collaborative filtering on un-partitioned data.

4.1 The Dataset

The data for this study is drawn from the EachMovie database (http://www.research.digital.com/SRC/eachmovie/), which consists of more than 2.8 million ratings on 1628 movie titles. We create a working set that consists of users who have rated at least 100 items. For these experiments we create a training set and test set by randomly selecting 80 and 20 ratings, respectively, from the rating history of each user in the working set.

4.2 The Offline/Online Execution Time

We present the performance of *RecTree* and *CorrCF* under the two regimes we call off-line and on-line operation. Off-line processing occurs when the systems require re-initialization. The systems train on the rating data and then compute predictions for all items that the user has yet to rate. When a user subsequently visits the systems, a lookup in constant time yields his recommendations.

In on-line processing the systems defer computing predictions; rather than exhaustively computing all recommendations, only a recommendation for the requested item is computed. This is quite often the case when a new user joins a recommendation service and a complete off-line processing may not be feasible.

The off-line and on-line execution time for *RecTree* as a function of the number of users and maximum partition size, *b*, is shown in **Fig. 3** and **Fig. 4**, respectively. The experiments were run 10 times for each partition size and the average running times reported. *RecTree* outperforms *CorrCF* for all partition and dataset sizes tested.

Like other memory-based collaborative filters, *CorrCF*'s quadratic off-line performance derives from its exhaustive search for advisors. In contrast, *RecTree* limits its search to within the partitions. As more users are added to the database, more partitions are crated to accommodate them. This strategy allows *RecTree* to scale by the number of clusters rather than the number of users.

RecTree's on-line performance is independent of the number of users already in the system. *RecTree* traverses its branches to locate the cluster closest to the new user and then takes a weighted aggregate of his advisors' rating to yield a recommendation. Since the cliques are approximately constant in size, the execution time is independent of dataset size. In contrast, *CorrCF* requires a scan of the entire database to aggregate all of the advisor ratings. As more users are added to the system, the cost of the scan increases.

RecTree's execution time improves with smaller partitions. Although the algorithm spends more time creating the cliques, this cost is more than offset by the savings in computing the similarity matrices. Smaller cliques however entail a reduction in the accuracy that we discuss in the next section.

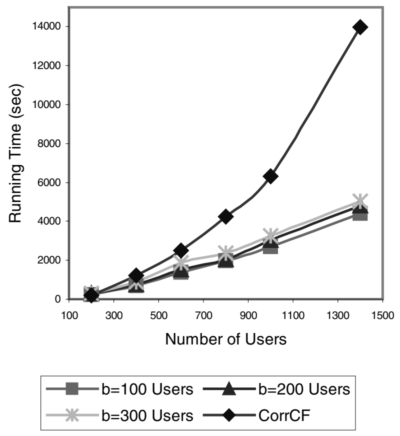


Fig. 3. Offline performance of *RecTree*.

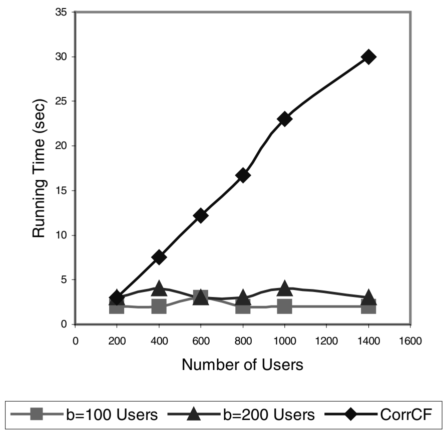


Fig. 4. On-line performance of *RecTree*.

4.3 The Accuracy Metric – NMAE

A popular statistical accuracy metric is the mean absolute error (MAE) [11] [14], which is the average absolute difference between the filter's recommendation and the user's actual vote. Goldberg et al. [4] proposes the normalized mean absolute error (NMAE), which normalizes the MAE by the rating scale. The NMAE has an intuitive explanation; it reflects the expected fractional deviation of predictions from actual ratings. The NMAE for a random filter applied to a random user, for example is 0.33 [4], which means that on average, we expect a prediction to be off by 33%. We use NMAE to report accuracy.

The accuracy of *RecTree* as a function of number of users and maximum partition size, *b*, is shown in **Fig. 5**; lower values of NMAE denote higher accuracy. *RecTree*'s improvement in accuracy with dataset size is typical of other collaborative filters. Larger datasets provide the CF algorithm with more candidates from which to select good advisors. The improvement however, is not necessarily monotonic; adding a batch of poorly correlated users will dilute the influence of existing good advisors. This dilution effect is clearly evident in the peak at 400 users; both *CorrCF* and *RecTree* for *b*<300 users show a drop in accuracy.

RecTree's accuracy is due in part to the success of the partitioning phase in localizing highly correlated users in the same partition. **Fig. 6** shows that the average similarity of advisors for *RecTree* is always higher than that of *CorrCF*. Because of the high intra-cluster similarity between users, *RecTree* is less susceptible to the dilution effect. At 1400 users in the dataset, *CorrCF* used an average of 223 advisors to compute a prediction. In contrast, *RecTree* for a partition size of 100 users, needed an average of only 46 highly correlated advisors.

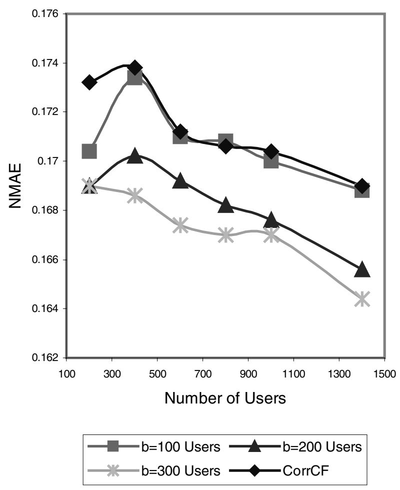


Fig. 5. NMAE of *RecTree* as a function of the number of users and partition size *b*.

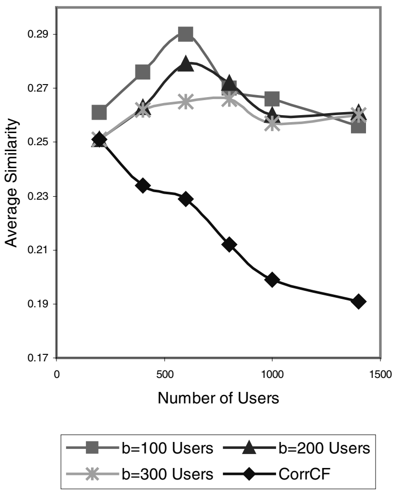


Fig. 6. The average similarity of advisors in *RecTree* exceeds *CorrCF*.

5 Discussion and Conclusion

In this paper, we describe, *RecTree*, a new linear CF method that applies clustering techniques to economically create cohesive cliques. *RecTree* achieves better scale-up in comparison to other memory based collaborative filters by seeking advisors only within a clique rather than the entire database. In particular, for off-line recommendation, *RecTree* scales by O($n\log_2(n)$) and O(b) for on-line recommendation, where n is the dataset size and b is the partition size, a constant.

RecTree achieves superior accuracy over *CorrCF* through default voting and the amelioration of the dilution effect. Default voting extends the rating history over which a similarity metric can be computed and hence improves the accuracy of a metric in capturing user proximity. The dilution effect is ameliorated by the high intra-partition similarity of a clique that acts as a coarse filter to limit the number of poor advisors that participate in computing a prediction.

Despite these improvements, we may yet be able to achieve higher accuracy by exploiting *RecTree*'s hierarchy of cliques. The current method computes predictions from only the members of the leaf cliques. We plan to investigate how the internal nodes of the *RecTree* data structure can contribute to even more accurate predictions.

The current implementation of *RecTree* assumes a single processor. However, the *RecTree* data structure can be easily distributed for parallel computation by a cluster of processors. For off-line processing, each branch of the tree can be grown independently of every other branch by assigning separate processing threads. For on-line processing, a thread can be assigned to each leaf node to handle prediction requests for new users. A parallel implementation of *RecTree* will be able to realize a greater throughput, which may be the subject of future work.

RecTree is an in-memory algorithm but for very large databases, it may not be an appropriate recommendation algorithm. It is the subject of future work to investigate the extension of *RecTree* to disk-resident databases with the incorporation of disk-based clustering algorithms, such as BIRCH [15].

References

[1] C. C. Aggarwal, C. Procopiuc, J. L. Wolf, P. S. Yu, and J. S. Park, Fast Algorithms for Projected Clustering, In *SIGMOD'99,* Philadephia, PA, June 1999.

[2] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan, Automatic Subspace Clustering in High Dimensional Data for Data Mining Applications, In *SIGMOD'98,* Seattle, WA, June 1998.

[3] J. S. Breese, D. Heckerman, and C. Kadie, Empirical analysis of predictive algorithms for collaborative filtering. In *Proc. 14th Conf. Uncertainty in Artificial Intelligence (UAI-98)*, pp. 43-52, San Franciso, CA, July 1998.

[4] K. Goldberg, T. Roeder, D. Gupta and C. Perkins, Eigentaste: A Constant Time Collaborative Filtering Algorithm, Information Retrieval, 2001.

[5] N. Good, J. B. Schafer, J. A. Konstan, A. Borchers, B. Sarwar, J. Herlocker and J. Riedl, Combining Collaborative Filtering with Personal Agents for Better Recommendations, In *AAAI-99,* July 1999 .

[6] S. Guha, R. Rastogi, and K. Shim, CURE: An Efficient Clustering Algorithm for Large Databases, In *SIGMOD'9),* pp. 73-84, Seattle, WA, June 1998.

[7] J. Han, S. Chee, and J. Y. Chiang, Issues for On-Line Analytical Mining of Data Warehouses, In *Proc. 1998 SIGMOD'96 Workshop on Research Issues on Data Mining and Knowledge Discovery (DMKD'98)* , Seattle, Washington, June 1998,

[8] J. L. Herlocker, J. A. Konstan, A. Borchers, and J. Riedl, An Algorithmic Framework for Performing Collaborative Filtering, In *Proc. 1999 Conf. Research and Development in Information Retrieval*, pp. 230-237, Berkeley, CA, August 1999.

[9] L. Kaufman and P. Rousseeuw, *Finding Groups in Data, An Introduction to Clustering Analysis.* John Wiley and Sons, 1989.

[10] J. A. Konstan, B. N. Miller, D. Maltz, J. L. Herlocker, L. R. Gordon, and J. Riedl, Applying Collaborative Filtering to Usenet News, *CACM*, 40(3): 77-87, 1997.

[11] P. Resnick, N. Iacovou, M. Sushak, P. Bergstrom, and J. Riedl, GroupLens: An open architechure for collaborative filtering of netnews. In *Proc. ACM Conf. Computer Support Cooperative Work (CSC) 1994,* New York, NY, Oct. 1994.

[12] B. M. Sarwar, J. A. Konstan, A. Borchers, J. L. Herlocker, B. N. Miller, and J. Riedl, Using Filtering Agents to Improve Prediction Quality in the Grouplens Research Collaborative Filtering System. In *Proc. ACM Conf. Computer Support Cooperativ Work (CSCW) 1998*, Seattle, WA., pp. 345-354 Nov. 1998.

[13] J. B. Schafer, J. Konstan, and J. Riedl, Recommender Systems in E-Commerce, *ACM Conf. Electronic Commerce (EC-99)*, Denver, CO, pp. 158-166, Nov. 1999.

[14] U. Shardanand and P. Maes, Social information filtering: Algorithms for automating "word of mouth." In *Proc. 1995 ACM Conf. Human Factors in Computing Systems,* New York, NY, pp. 210-217, 1995

[15] T. Zhang, R. Ramakrishnan, and M. Livny, "BIRCH: an efficient data clustering method for very large databases", In SIGMOD'96, Montreal, Canada, pp. 103-114, June 1996.

Discovering Web Document Associations for Web Site Summarization

K. Selçuk Candan[1]* and Wen-Syan Li[2]

[1] Computer Sci. and Eng. Dept, Arizona State University, Tempe, AZ 85287, USA
[2] CCRL, NEC USA, Inc., 110 Rio Robles M/S SJ100, San Jose, CA 95134, USA

Abstract. Complex web information structures prevent search engines from providing satisfactory context-sensitive retrieval. We see that in order to overcome this obstacle, it is essential to use techniques that recover the web authors' intentions and superimpose them with the users' retrieval contexts in summarizing web sites. Therefore, in this paper, we present a framework for discovering implicit associations among web documents for effective web site summarization. In the proposed framework, associations of web documents are induced by the web structure embedding them, as well as the contents of the documents and users' interests. We analyze the semantics of document associations and describe an algorithm which capture these semantics for enumerating and ranking possible document associations. We then use these asociations in creating context-sensitive summaries of web neighborhoods.

1 Introduction

Hypermedia has emerged as a primary means for storing and structuring information. This is primarily visible in continuously proliferating web based information infrastructures in corporate and e-commerce organizations. Yet, due to the continuously increasing size of these infrastructures, it is getting ever difficult for users to understand and navigate through such sites. Furthermore, these complex structures prevent search engines from providing satisfactory context-sensitive retrieval functionalities. We see that in order to overcome this obstacle, it is essential to use techniques that recover the web authors' intentions and superimpose it with the users' retrieval contexts. Therefore, in this paper, we present a framework for creating web site summarizations.

When an author or a web designer prepares a web document, he/she would put intended information not only on in the textual content of a page; but, also on the link structure of the web site. Thus, the content of web pages along with the structure of the web domain can be used to derive hints to summarize web sites. What differentiates our work from similar work in the literature is that, we propose to use *associations between documents in a neighborhood and the reasons why they are associated* in the summarization process. Knowing these reasons, among other things, is essential in (1) in creating web site maps that

* This work was performed when the author visited NEC, CCRL.

matches web designers' intentions and (2) in superimposing the logical structure of a web site with the context provided by a visitors interests.

In the next section, we describe how to construct of meaningful summarizations of web neighborhoods, which can be viewed and used as a web site map. In Section 3, we present how to mine web document associations using page content as well as link structures. In 4, we describe the experiment results for our web site summarization algorithms, respectively. Finally we conclude this paper with discussion of future work.

2 Web Site Summarization

In this section, we introduce the web site summarization task and discuss how to use document associations for this purpose. We see that corporate sites, and most of the web space, is composed of two types of neighborhood: physical and logical. The physical neighborhoods are separated from each other through domain boundaries or directory names. The logical neighborhoods, on the other hand, are mostly overlapping and they can cover multiple domains or they can be limited to a part of a single domain. Web designers usually aim at overlapping the physical neighborhoods of the web site with the logical neighborhoods. However, as (1) the foci of users may differ from each other, (2) the focus of a single user may shift from time to time, such a strict design may loose its intended effect over time.

Physical neighborhoods are usually decided by the URL structures of the web sites. In [1], we described algorithms to discover logical neighborhoods. Consequently, in this section, we will assume that a corporate web site, W, is already partitioned into its neighborhoods. We denote this partitioning as a partially ordered lattice $\mathcal{W}$:

- each vertex w_i in $\mathcal{W}$ is a set of nodes (or a neighborhood) and
- if w_j is a child of w_i, then w_j is a sub-neighborhood of w_i; furthermore, $w_i \cap w_j = \{v_{ij}\}$, where v_{ij} is the entry point to sub-neighborhood w_j from neighborhood w_i (Figure 1(a)).

Intuitively, the lattice corresponds to a hierarchy of neighborhoods. At the highest level, we have a neighborhood which consists of high-level corporate pages and the entry pages of lower neighborhoods. Similarly, each neighborhood consists of a set of high-level pages and the entry-pages of all its sub-neighborhoods. Consequently, summarization of W involves of two tasks: (1) identification of which nodes in the partially ordered lattice $\mathcal{W}$ will be shown to the user (i.e., focusing on the neighborhoods) and (2) summarization of each focussed neighborhood based on user interest (Figure 1(b)). Below, we discuss these to tasks in greater detail:

2.1 Identification of Focus Neighborhoods

In order to identify the focus neighborhoods, we need to start from the root neighborhood, w_1 of $\mathcal{W}$. This neighborhood contains, the high-level pages of the

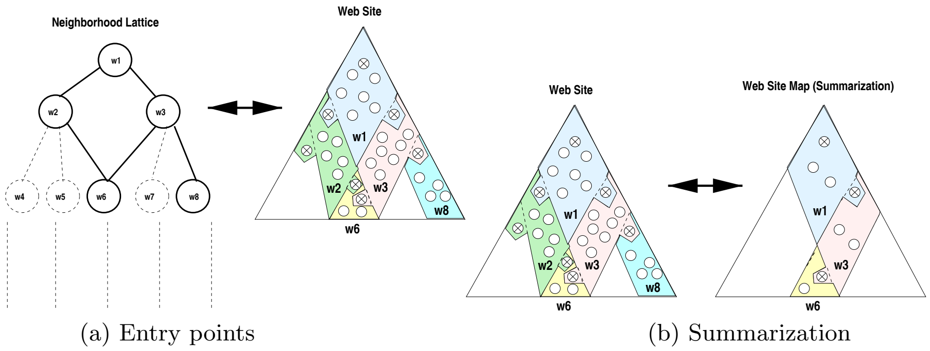


Fig. 1. The crossed circles denote the entry points of neighborhoods. Each sub-neighborhood contain one entry point per its parent neighborhoods. Each parent neighborhood includes the entry points of its sub-neighborhoods.

given corporate site along with the entry level pages of it sub-neighborhoods. Let us assume that we are interested in a predetermined number, k, of focus points in this top neighborhood. Then, we could rephrase the focus identification problem in terms of the neighborhood summarization problem:

– In order to identify the k focal points of the neighborhood w_1 of $\mathcal{W}$, summarize w_1 into a graph of size k. The k remaining pages are in user focus.
– Let us assume that $F = \{v_1, v_2, \ldots, v_k\}$ are the k pages in the summary of w_1. If $v_i \in F$ is an entry-page of a sub-neighborhood, w_i, then repeat the same process for the sub-neighborhood w_i.

Note that the above recursive process allow us to identify the focal points of a web site. The parameter k is user-dependent and describes how focussed the user would like to be: smaller values of k correspond to more focussed site maps. Note also that, this recursive algorithm assumes that given a neighborhood, we can create a summary of size k. Next, we describe how we can achieve this task.

2.2 Summarization of a Neighborhood

Each neighborhood, $w_i \in \mathcal{W}$ consists of a set of neighborhood pages and the entry-pages of its sub-neighborhoods. Also, from each of its parent-neighborhoods, w_i is reached through one entry-page (Figure 1). Let us assume that the set, $\mathcal{E}$, of pages correspond to the entry pages of w_i from all its parent-neighborhoods that are in focus. Then our goal is to summarize the neighborhood with respect to these entry points as well as the content-description provided by the user. The summarized neighborhood will give the focussed pages in this neighborhood and the corresponding connectivity.

In order to summarize a given neighborhood, we first have to identify the pages that are important. In this case, the entry pages of a neighborhood (from parents in focus) are relatively important as they will connect the web site maps of the neighborhoods. Note also that the entry pages of the sub-neighborhoods are also important as they will extent the map downwards in the hierarchy, given that the lower neighborhoods are also in focus.

Therefore, given a neighborhood, w_i, the set, $\mathcal{E}$, of focussed entry pages from its parents, and the set, $\mathcal{L}$, of entry-pages to its sub-neighborhoods, we can create a set of seed pages (for summary) $\mathcal{S} = \mathcal{E} \cup \mathcal{L}$. Then our goal is,

– given the set, $\mathcal{S}$, of seed (entry) web pages,
– potentially a content-description,
– a web neighborhood, $G^N = w_i$ which contains these seeds, and
– an integer k,

to create a *summary*, with k pages, of the neighborhood with respect to the seed pages.

Observation: Since, the web site map is a set of representative nodes in a web site, the nodes in a web site map needs to satisfy the following criteria:

– High connectivity so that users can navigate from these web site map nodes to other nodes easily.
– The contents of these web site map nodes need to be representative.

Thus, the nodes selected to form a web site map need to be both structural and content-wise representative for all pages in a web site. Therefore, we can use the selected nodes, *which describe the association between the pages in the site,* as a summary of a web site and to form a site map.

Therefore, given a graph $G(V, E)$, its undirected version $G^u(V, E^u)$, and k *dominant vertices* in V with respect to the given seed vertices $\mathcal{S}$, we can construct a k-summary $\Sigma^k_{(\mathcal{S})}(V^\sigma, E^\sigma, \delta)$ of the input graph (δ is a mapping, $E^\sigma \rightarrow R^+ \times \{left, right, bi, none\}$, which describes the lengths and directions of the summarized edges) as shown in Figure 2.

Note that Step 3 of the algorithm requires the identification of the k most dominant vertices (or the vertices which describe the document associations the best) in the graph with respect to the seed vertices. These vertices will be the only vertices used in the summarization (Step 4). Given two dominant vertices, v_i and v_j, (to be visualized in the summary) Step 5 of the algorithm first constructs a temporary graph, $G_{temp}(V_{temp}, E_{temp})$, from the original web graph, such that no path between v_i and v_j can pass through another dominant node. Then, it uses the shortest path, $sp(v_i, v_j)$, between v_i and v_j in this temporary graph to identify edges that are to be visualized to the user. Consequently, a given edge that is included in the summary denote the shortest path, between two dominant vertices, that do not pass through other dominant vertices. Hence, this step eliminates the possibility of inclusion of redundant edges.

Note that, for the identification of shortest paths to be visualized, the path length can be defined in various ways. One possibility would be *minimization of the number of edges on the path.* This would be useful when the aim is to

```
1. Let $G^N(V^N, E^N) \subseteq G^u$ be the neighborhood graph;
2. $V^\sigma = \emptyset$; $E^\sigma = \emptyset$;
3. Let $\mathcal{K} \subseteq V_{G^N}$ be the set of $k$ vertices with the highest dominance values;
4. $V^\sigma = \mathcal{K}$;
5. For each $v_i$ and $v_j \in V^\sigma$
   a) $V_{temp} = V^N - \{V^\sigma - \{v_i, v_j\}\}$;
   b) $E_{temp} = \{\langle v_k, v_l\rangle \mid (\langle v_k, v_l\rangle \in E^N) \wedge (v_k \in V_{temp}) \wedge (v_l \in V_{temp})\}$
   c) If $sp(v_i, v_j)$ is the shortest path in $G_{temp}(V_{temp}, E_{temp})$ between $v_i$ and
      $v_j$ then
      i. Let the length of the path $sp(v_i, v_j)$ be $\Delta$
      ii. $E^\sigma = E^\sigma \cup \{e = \langle v_i, v_j\rangle\}$;
      iii. If $v_j$ is reachable from $v_i$ in the directed graph $G$ through the vertices
           in $sp(v_i, v_j)$, but if $v_i$ is not reachable from $v_j$, then $\delta(e) = \langle \Delta, right\rangle$

      iv. If $v_i$ is reachable from $v_j$ in the directed graph $G$ through the vertices
          in $sp(v_i, v_j)$, but $v_j$ is not reachable from $v_i$, then $\delta(e) = \langle \Delta, left\rangle$
      v. If $v_i$ and $v_j$ are reachable from each other in the directed graph $G$
         through the vertices in $sp(v_i, v_j)$, then $\delta(e) = \langle \Delta, bi\rangle$
      vi. If neither $v_i$ nor $v_j$ is reachable from the other in the directed graph
          $G$ through the vertices in $sp(v_i, v_j)$, then $\delta(e) = \langle \Delta, none\rangle$
```

Fig. 2. Algorithm for constructing a summary

display the user information about the connectivity of the summarized graph. Once the edges to include in the summary, the sub-steps of Step 5(c) gathers more information regarding each edge (such as the reachability of the two pages at the end-points from each other in the inherently directed web) and reflects this information to the summary in the form of edge labels.

Example 1. Let us consider the graph in Figure 3(a). and let us assume that we are asked to construct 5- and 7-summaries of it. For this example, let us also assume that after running an association mining algorithm, we have found that the seven dominant vertices in this neighborhood (relative to a set of seed vertices) are A, B, F, E, J, C, and I. For simplicity, let us also assume that the length of the path is defined using the number of edges on it. Figures 3(b) and (c) shows the corresponding 5- and 7-summaries of this graph. The labels of the edges denote the length of the corresponding paths and the arrows on the edges denote the reachability of the end-points from each other (e.g., an edge of the form "$\longleftrightarrow$" denotes that both end-points can reach the other one over the web through this shortest path; whereas, an edge of the form "—" denotes that neither of the end-points can reach the other one over the web through this shortest path.

Those entry pages which are still in the map after the summarization are called *focussed entry-pages*, and they point to the other logical domains that have to be further explored and summarized. Therefore, we recursively apply the summarization algorithm described above for those domains who have at least one *focused entry-page.*

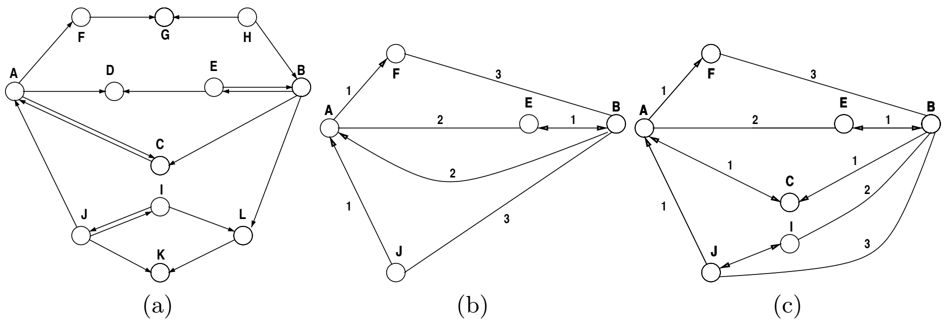


Fig. 3. (a) The web neighborhood in our running example, (b) its web site map with 5 nodes, and (c) its web site map with 7 nodes

3 Finding Document Associations

We now use an example to illustrate the task of idenifying document associations and overview of our approach.

Example 2. In Figure 4, we show a set of links between two web pages `W.Li` and `D.Agrawal` (both are highlighted in the figure). The purpose of each web link is indicated as its label in the figure. For example, `W.Li` graduated from Northwestern University; whereas `D.Agrawal` and `P.Scheuermann` both graduated from SUNY. Based on this link structure, if we want to find the association between `W.Li` and `D.Agrawal` pages, the link structure connecting `W.Li` and `D.Agrawal` are useful clue for mining. Below, we enumerate some associations that are embedded in this graph:

– *Association 1:* `Web8 paper` page appears in a path of distance of 2 connecting the pages `W.Li` and `D.Agrawal`. Therefore, `W.Li` and `D.Agrawal` may be associated due to a co-authored paper.
– *Association 2:* `Y.Wu` page is on two paths related to NEC Research Laboratories, each of distance 4. `W.Li` and `D.Agrawal` may be associated due to the fact they both supervised `Y.Wu` at different occasions or they participate in the same project at NEC.
– *Association 3:* `WOWS'99` and `ACM DL'99` pages appear on a single path of distance 3. Such an association can be interpreted as that `W.Li` and `D.Agrawal` are participating in the same conference (e.g. presentation or program committee members).

The above example shows that the following two intuitions, along with the actual content of the pages, can generally be used to identify why a given set of pages are associated:

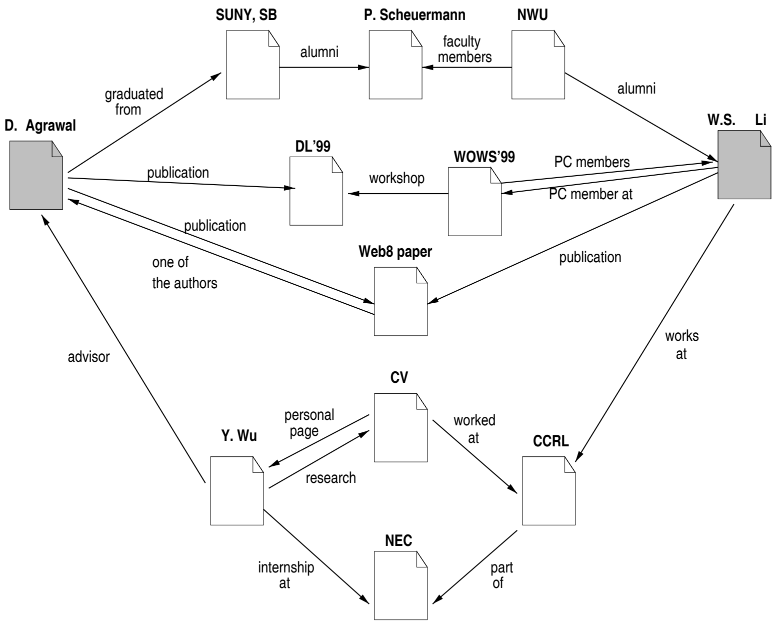


Fig. 4. Link structure associating the web documents `W.Li` and `D.Agrawal`

1. Pages on a shorter path between the `W.Li` and `D.Agrawal` pages are stronger indicators than others to reflect why the `W.Li` and `D.Agrawal` pages are associated.
2. Pages which appear on more paths should be stronger indicators than others to reflect why the `W.Li` and `D.Agrawal` pages are associated.

Note that a page with a higher connectivity (i.e. more incoming links and outgoing links) is more likely to be included in more paths; consequently, such a page is more likely to be ranked higher according to the above criteria. This is consistent with the principle of topic distillation[2,3]. On the other hand, a page with a high connectivity but far away from the seed URLs may be less significant to represent the associations than a page with low connectivity but close to the seed URLs. A page which satisfies both of the above criteria (i.e. near seed URLs and with high connectivity) would be a good representative for the association.

Obviously, the *distance* between two pages can be defined in various ways. In the simplest case, the number of links between two pages can be used as the distance metric. On the other hand, in order to capture the physical as well as logical distances between pages, we use different distance metrics capable of capturing document contents as well as user interests.

Based on this motivation, we use a novel framework for mining associations among web documents using information *implicitly* reflected in the links connecting them, as well as *the contents* of these connecting documents. We develop a web mining technique, based on a *random walk algorithm*, which considers three factors: (1) document distances by link; (2) connectivity; and (3) document content. Consequently, instead of explicitly defining a metric, we choose a set of random walk parameters that will implicitly capture the essence of these observations. The details and the complexity of this algorithm has been analytically and experimentally studied in [4]. In the next section, we present the experimental results on Web site summarization.

3.1 Comparison with Other Approaches

Note that the well-known algorithm *topic distillation* [2,5,6] could be a natural choice for summarization purposes. However, we observe that the behavior of the topic distillation algorithm may not be as good as our *document association* based approach in the scope of summarization tasks. The reason is that the topic distillation algorithm aims at selecting a small subset of the most "authoritative" pages and "hub" pages from a much larger set of query result pages. An authoritative page is a page with many incoming links and a hub page is a page with many outgoing links. Such authoritative pages and hub pages are mutually reinforcing: good authoritative pages are linked by a large number of good hub pages and vice versa. Because the important pages are mutually reinforcing, the results tend to form a cluster. For example, there are many on-line HTML papers at `www-db.stanford.edu`. These papers consist of a number of pages linking to each other. By using the topic distillation algorithm, most of these pages are selected while many individual home pages are left out; this is not suitable for the purposes of summarization. On the other hand, our algorithm uses the concept of *seed nodes* which focus the summarization process, based on the web site structure as well as the user focus. Nodes selected after summarization are those nodes that explain why the seed nodes are related. Therefore a good choice of seed nodes (in our case, the entry-nodes of logical domains) leads into a good and meaningful summarization. Thus, a document association based approach is more suitable for the purpose of summarization. Researchers in the AI community have developed Web navigation tour guides, such as WebWatcher[7]. WebWatcher utilizes user access patterns in a particular Web site to recommend users proper navigation paths for a given topic. User access paterns can be incorporated into the random walk-based algorithm to improve the document association mining.

4 Experiments on Web Site Map Generation

We have conducted a set of experiments on *www-db.standford.edu*, which has 3040 pages and $12,581$ edges. The average number of edges per pages is 3.5. The

Table 1. Summarization results: (a) root domain and (b) a subdomain

(a)

Score	URL
0.124	/LIC/LIC.html
0.110	/LIC/mediator.html
0.032	/people/
0.031	/~gio/
0.018	/~wangz/
0.018	/~jan/watch/intro.html
0.016	/tsimmis/
0.016	/~danliu/
0.015	/cs347/
0.015	/LIC/
0.015	/~widom/
0.014	/~wilburt/
0.014	/~chenli/
0.013	/~ullman/
0.012	/CHAIMS/
0.012	/~echang/
0.012	/~cyw/
0.012	/~crespo/
0.012	/~cho/
0.012	/~sergey/

(b)

Score	URL
0.015	/pub/gio/CS545/image.html
0.011	/pub/gio/1999/Interopdocfigs.html
0.011	/pub/gio/biblio/master.html
0.011	/pub/gio/CS99I/library.html
0.010	/pub/gio/1994/vocabulary.html
0.009	/pub/gio/CS99I/ubi.html
0.009	/pub/gio/CS99I/entedu.html
0.009	/pub/gio/CS99I/health.html
0.008	/pub/gio/CS99I/wais.html
0.007	/pub/gio/CS99I/security.html
0.006	/pub/gio/CS99I/refs.html
0.005	/pub/gio/gio-papers.html
0.005	/pub/gio/inprogress.html
0.005	/pub/gio/
0.003	/pub/gio/paperlist.html
0.003	/pub/gio/CS99I/description.html
0.003	/pub/gio/CS99I/background/Cairn......
0.003	/pub/gio/CS545/
0.003	/pub/gio/CS99I/copyright.html
0.002	/pub/gio/CS545/indexing/

experiments were ran on a 500MHz Pentium Architecture Linux OS PC with 128 MB of RAM. Using this setup, we have conducted a set of experiments to validate the web site summarization algorithm presented in this section. Here, we report on the main findings using one example case:

We asked our system to summarize the *www-db.standford.edu* domain, with respect to a context defined as "publication or paper". We also asked our system to give higher importance to more recently updated pages. First of all, our system identified 42 logical domains among 3040 pages. Then, we used the algorithm described in this paper, to recursively summarize this logical structure with respect to the defined context.

The algorithm started by summarizing the root logical domain which consists of all the entry-pages of the logical domain in the second level and the all the page in root logical domain, where *www-db.stanford.edu* is the entry page. Thus, 1584 pages are included for the experiments. The result of this summarization, using a radius of 2, is shown in Table 1(a) (we omit the summarized edge to simplify the discussion). Note that most of these pages are actual home pages and entry pages to the lower level logical domains. This was due to the fact that only these home pages are relevant to the focused keywords *papers* and *publications* and their edges are assigned with a lower cost. Thus, the algorithm prefers to "walk" through these pages over other 1400 pages. Some pages, such as `www-db.stanford.edu/LIC/` and `www-db.stanford.edu/LIC/mediator.html`, in the root logical domain are included due to its high connectivity.

Next, the algorithm recursively visited the nodes in this domain. Here we report on one of the largest subdomains, i.e., *http://www-db.stanford.edu/˜gio/*, (793 pages). When summarized with the proposed algorithm, the resulting graph contained the pages shown in Table 1(b). Note that these pages are either publication oriented pages, or they are linked to many pages with publication content: actual publication pages are omitted from the summarization to give place to pages which connect to many publication pages, allowing easier access to more information while browsing the web.

5 Conclusion

Hypermedia has emerged as primary means for structuring documents and for accessing the web. In this paper, we present a framework for site map construction and web page summarization. For this purpose, we introduce a random walk algorithm for mining implicit associations among web documents, induced by Web link structures and document contents. Link information has been used by many search engines to rank query results as well as finding relevant documents and web sites. We compare and contrast our random walk algorithm with other existing work, such as various topic distillation techniques.

References

[1] Wen-Syan Li, Okan Kolak, Quoc Vu, and Hajime Takano. Defining Logical Domains in a Web Site. In *Proceedings of the 11th ACM Conference on Hypertext*, pages 123–132, San Antonio, TX, USA, May 2000.

[2] Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, pages 668–677, January 1998.

[3] Wen-Syan Li and K Selçuk Candan. Integrating Content Search with Structure Analysis for Hypermedia Retrieval and Management. *ACM Computing Surveys*, 31(4es):13, 1999.

[4] K. Selçuk Candan and Wen-Syan Li. Using Random Walks for Mining Web Document Associations. In *Proceedings of the 4th Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 294–305, Kyoto, Japan, April 2000.

[5] Krishna Bharat and Monika Henzinger. Improved algorithms for topic distillation in a hyperlinked environment. In *Proceedings of the 21th Annual International ACM SIGIR Conference*, pages 104–111, Melbourne, Australia, August 1998.

[6] Lawrence Page and Sergey Brin. The Anatomy of a Large-Scale Hypertextual Web Search Engine. In *Proceedings of the 7th World-Wide Web Conference*, Brisbane, Queensland, Australia, April 1998.

[7] T. Joachims, D. Freitag, and T. Mitchell. Webwatcher: A tour guide for the world wide web. In *Proceedings of the 1997 Internaltional Joint Conference on Artificial Intelligence*, August 1997.

Data Visualization and Analysis with Self-Organizing Maps in Learning Metrics

Samuel Kaski, Janne Sinkkonen, and Jaakko Peltonen

Neural Networks Research Centre, Helsinki University of Technology,
P.O. Box 5400, FIN-02015 HUT, Finland
{samuel.kaski, janne.sinkkonen, jaakko.peltonen}@hut.fi
http://www.cis.hut.fi

Abstract. High-dimensional data can be visualized and analyzed with the Self-Organizing Map, a method for clustering data and visualizing it on a lower-dimensional display. Results depend on the (often Euclidean) distance measure of the data space. We introduce an improved metric that emphasizes important local directions by measuring changes in an auxiliary, interesting property of the data points, for example their class. A Self-Organizing Map is computed in the new metric and used for visualizing and clustering the data. The trained map represents directions of highest relevance for the property of interest. In data analysis it is especially beneficial that the importance of the original data variables throughout the data space can be assessed and visualized. We apply the method to analyze the bankruptcy risk of Finnish enterprises.

1 Introduction

Visualization of complex, multidimensional data sets is a fundamental part of exploratory data analysis. There exist a wealth of alternative methods from icon-based techniques to projection techniques (see e.g. [13,3]). In this paper we use the Self-Organizing Map [16,17], a method that has good properties of both projection and clustering methods: it reduces the dimensionality of a data set by visualizing it on a lower-dimensional display, and reduces the amount of data by representing them with a smaller number of models ordered on a discrete map lattice.

We present a solution to a particular problem in data exploration: The results of all visualization methods depend on the original representation of the data, i.e., the variables or features chosen to describe the data objects, and their relative scaling. The representation is not unique, and the analyst must decide which features of the data are important for the analysis. The solution we present is a measure of similarity, a metric, of the data space that removes the nonuniqueness and concentrates on measuring the important changes (to be made exact later) in the data. The new metric is particularly well-suited for the Self-Organizing Maps but should be useful at least for other kinds of projection methods.

In some tasks there exists additional, indirect information that can be used to fix the representation. Here we study one such setting, that of multivariate continuous data, the primary data, associated with a discrete variable, the auxiliary data. We assume that the discrete variable is so fundamental for the problem at hand that all the *interesting* variation of the real-valued data is reflected in the discrete variable. In this paper we will define distances in the primary data space in terms of how much the interesting auxiliary data changes.

Technically, we will estimate the changes in the auxiliary data by an estimator of its conditional density given the primary data, and derive a local distance measure based on the Fisher information matrix. A Self-Organizing Map (SOM) will be trained in this new metric. Then the SOM will only describe variation which is meaningful from the viewpoint of the interesting, auxiliary variable.

In a case study, we will visualize the variation of financial indicators of Finnish companies that is associated to their bankruptcy risk, a task into which SOMs in Euclidean metrics have already been applied ([14],[15]). The SOM will be used to graphically display essential properties of the data set, namely the distribution of bankruptcy risk among the companies together with their characterization in terms of the financial indicators. It will also be used in visualizations of the factors affecting the bankruptcy sensitivity of different kinds of companies. Our methods are general in the sense that they can be applied to any kinds of multivariate data with an associated discrete variable.

2 The Self-Organizing Map

The Self-Organizing Map (SOM) [16,17] is a tool that can be used for creating *overviews* of data sets. The SOM can be visualized as a graphical display on which the data are projected non-linearly in an ordered fashion: close-by locations represent data that are similar in the multidimensional data space. The order of the data items on the map then reveals their mutual similarities that have been "hidden" in the possibly large amount of high dimensional data. The same display can be used for visualizing additional properties of the data such as its clusteredness and the distribution of the values of the original data variables. Over 4000 scientific papers have already been written on the SOM (`http://www.cis.hut.fi/research/som-bibl/`; cf. [11]). Here we will concentrate on describing only the properties most relevant to our work on metrics and visualizations. The starting point of SOM-based data analysis is a set of multivariate continuous-valued data items (equivalently, observations or records), denoted by the data vectors $\mathbf{x}_k$, $k = 1, \ldots, N$. The SOM is a regular map lattice on which a *model vector* $\mathbf{m}_i$ is attached at regularly spaced positions. An example where the models have been attached onto a hexagonal two-dimensional lattice is shown in Figure 1. The dimensionality of the model vectors and the data vectors is the same, and for visualization purposes the lattice is usually chosen to be two-dimensional.

The data set can be visualized on the SOM display by projecting each sample $\mathbf{x}_k$ onto the location of the map lattice that corresponds to the closest model vector. As a result of the SOM algorithm the set of the model vectors represents the distribution of the data in an ordered fashion: model vectors that are close-by on the map lattice are close-by in the data space as well. Therefore also the projection of the data on the map is similarly ordered. The two-dimensional map lattice can intuitively be thought of as an "elastic network" that is being fitted to the input data. Each model vector determines the location of the corresponding node of the network in the input data space. The network tries to follow the distribution of the input data while remaining organized in the sense that model vectors at neighboring positions on the map lattice are close to each other.

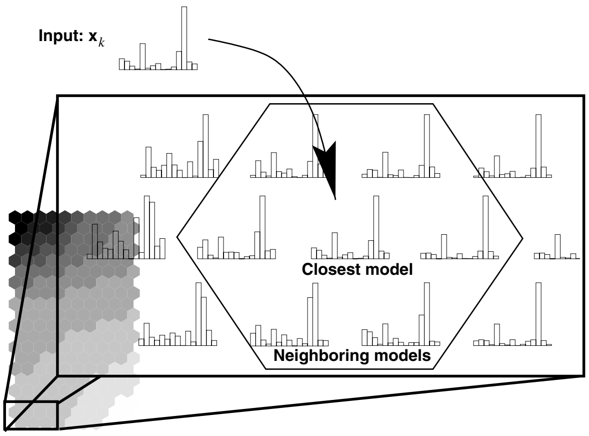


Fig. 1. The Self-Organizing Map consists of a set of model vectors, depicted here as bar graphs where each bar corresponds to one component. The models are attached onto a *map lattice* at regularly spaced intervals. A sample map display is shown in the lower left corner, and the enlarged portion shows the map lattice that underlies the bottom left corner of the map display. The arrow illustrates the projection of the input vector $\mathbf{x}_k$ to the SOM, and the hexagon denotes the neighborhood of the closest model.

In the stochastic version of the SOM algorithm the model vectors are estimated in an iterative process. At each step of the iteration one data vector $\mathbf{x}$, i.e., a multidimensional observation, is chosen with replacement. The closest model vector, defined by the expression

$$\|\mathbf{x} - \mathbf{m}_c\| \leq \|\mathbf{x} - \mathbf{m}_i\| \;\; \forall i \; , \tag{1}$$

is then searched for. Here $c = c(\mathbf{x})$ indexes the closest model vector. The norm is usually Euclidean; in Section 4 we will introduce a new kind of a metric.

After the closest model has been found it is modified towards the current input to make the model represent the input better. When this process is iterated, each time randomly choosing a new observation, the different model vectors gradually begin to specialize in representing different types of observations.

The algorithm described so far is essentially a stochastic version of the well-known K-means clustering approach. What distinguishes the SOM from the K-means is that in addition to the closest model *the neighboring models are* *updated as well.* The neighborhoods are defined in terms of closeness *on the map lattice* as demonstrated in Figure 1. The update of the neighboring models can be expressed in a general form by introducing a *neighborhood kernel* h_{ci} which is a decreasing function of the distance between the lattice points i and c. The kernel determines how much the model vector indexed with i will be modified when the model vector c is closest to the current observation $\mathbf{x}$. Using this notation the modification at step t of the stochastic iteration can be expressed as

$$\mathbf{m}_i(t+1) = \mathbf{m}_i(t) + h_{c(\mathbf{x}),i}(t)\left(\mathbf{x}(t) - \mathbf{m}_i(t)\right) . \tag{2}$$

When the iterative process defined by the equations (1) and (2) is continued, model vectors that are attached to neighboring positions on the map lattice gradually become similar. The process forms a neighborhood preserving mapping of the lattice into the input space.

During the stochastic approximation the length of the step, $\alpha(t)$, decreases gradually towards zero. In equation (2) the length is incorporated into the neighborhood function $h(t) = \alpha(t)h'(t)$.

To guarantee global organization of the map the "width" of h will be made to decrease as well. It is advisable to start the learning with a wide neighborhood function which narrows rapidly. After that the final values of the model vectors can be estimated using a narrower neighborhood function.

More details of alternative versions of the SOM and other ways of computing the model vectors can be found in [17].

3 The Learning Metric

Assume now that there is auxiliary data available about the primary data $\mathbf{x}$. It is assumed that changes in the auxiliary data are important, and hence the factors or directions in the primary space that correspond to the changes are potentially important as well. In this section we will derive a metric that measures distances in the primary data space as changes in the auxiliary data. Technically speaking, the set of available data consists of paired values of two random variables. Our aim is to extract the variation of continuous-valued multivariate data $\mathbf{x}_k$ which affects another, discrete variable with values c_k. In our case study of bankruptcies, c_k is binary-valued and indicates whether an enterprise goes bankrupt within the next three years, and the $\mathbf{x}_k$ are feature vectors derived from financial statements.

To operationally define the variation of $\mathbf{x}$ affecting c, we need a model of their mutual dependence. Here we have used a Gaussian mixture model to estimate the conditional probabilities $p(c|\mathbf{x})$ (details in the end of this Section).

The model of the conditional probabilities will be used for measuring the effect of the variation of $\mathbf{x}$ on c. The amount of variation will be the distance measure (metric) of the data space.

In principle the distance between a pair of input samples $\mathbf{x}$ and $\mathbf{x}'$ could be measured by the difference between the distributions of c, given $\mathbf{x}$ and $\mathbf{x}'$. Such a distance measure might be sometimes useful, but it has two disadvantages. First, if based on the Kullback-Leibler divergence, it is not symmetric and therefore not a metric. Second, in such a metric all points having similar conditional distribution would be close to each other irrespective of where they are in the original data space. Different factors may, however, be important at different locations of the data space. Often we are interested in measuring *local* changes, for example measuring which factors most affect the bankruptcy probability of a certain kind of a company.

Because of these drawbacks of the straightforward application of the Kullback-Leibler divergence, and because it will turn out that only local distances are really needed in computing a SOM, we resort to measuring distances by local changes of the estimates $\hat{p}(c|\mathbf{x})$. The differentiability of $\hat{p}(c|\mathbf{x})$ with respect to $\mathbf{x}$ is naturally assumed.

A classic result [18] states that the local Kullback-Leibler distances can be measured using the Fisher information matrix. We *define* the (squared) distance by the localized Kullback-Leibler distance, expressed as

$$d^2(\mathbf{x}, \mathbf{x} + d\mathbf{x}) \equiv D(\hat{p}(c|\mathbf{x}) \| \hat{p}(c|\mathbf{x} + d\mathbf{x})) = d\mathbf{x}^T \mathbf{J}(\mathbf{x}) d\mathbf{x} , \tag{3}$$

where

$$\mathbf{J}(\mathbf{x}) = E_{\hat{p}(c|\mathbf{x})} \left\{ \left(\frac{\partial}{\partial \mathbf{x}} \log \hat{p}(c|\mathbf{x}) \right) \left(\frac{\partial}{\partial \mathbf{x}} \log \hat{p}(c|\mathbf{x}) \right)^T \right\} \tag{4}$$

is the Fisher information matrix and $E_{\hat{p}(c|\mathbf{x})}$ denotes expectation over the possible values of C, conditioned on $\mathbf{x}$. (The local distance measure is sufficient for our purposes. It can be extended into a global Riemannian metric, defined as minimal path integrals of the local distances [1,12,19]. The computation of such path integrals is, of course, non-trivial.)

Note that the Fisher information matrix was originally defined to measure how changing model parameters affects the probability distribution produced by the model [20]. Here, in contrast, we measure how a movement in the data space affects the conditional probabilities $\hat{p}(c|\mathbf{x})$. Hence $\mathbf{x}$ has the role originally given to model parameters. In the new metric, all movements of equal (infinitesimal) length produce equally large effects in $\hat{p}(c|\mathbf{x})$ as measured by the Kullback-Leibler divergence.

3.1 Metrics from a Mixture Density Model

For estimating $p(c|\mathbf{x})$, plenty of alternative methods are available (for reviews see e.g. [8,21]). In this paper we do not search for the best possible estimator but resort to the Mixture Discriminant Analysis 2 [6] (MDA2; cf. also [7]), a version of the classic generative Gaussian mixture model that originally estimates the joint density of a discrete and a continuous multidimensional variable. The model is

$$\hat{p}(c_i, \mathbf{x}) = \sum_j \pi_j \xi_{ji} b_j(\mathbf{x}; \boldsymbol{\theta}_j) \tag{5}$$

where the b_j are spherically symmetric Gaussian basis functions of a common width σ, and the π_j and ξ_{ji} are parameters with $\sum_j \pi_j = 1$ and $\sum_i \xi_{ji} = 1$. By applying the Bayes rule, we obtain an estimate of the conditional density $p(c|x)$:

$$\hat{p}(c_i|\mathbf{x}) = \frac{\sum_j \pi_j \xi_{ji} b_j(\mathbf{x}; \boldsymbol{\theta}_j)}{\sum_j \pi_j b_j(\mathbf{x}; \boldsymbol{\theta}_j)} \; . \tag{6}$$

The model (5) is a mixture of component probabilities $\xi_{ji} b_j(\mathbf{x}; \boldsymbol{\theta}_j)$ that correspond to Gaussian distributions, one of which is picked to generate each data sample. It can be shown that under this model the distances (3) become

$$\sigma^4 d^2(\mathbf{x}, \mathbf{x} + d\mathbf{x}) = E_{\hat{p}(c|\mathbf{x})} \left\{ \left[d\mathbf{x}^T \left(E_{p(u_j|\mathbf{x}, c_i; \boldsymbol{\theta}_j)}\{\boldsymbol{\theta}_j\} - E_{p(u_j|\mathbf{x}; \boldsymbol{\theta}_j)}\{\boldsymbol{\theta}_j\} \right) \right]^2 \right\} \; , \tag{7}$$

where the u_j are values of a multinomial random variable U that indicates which of the components in the mixture has generated $\mathbf{x}$ and c. The parameter σ governs the width of the Gaussians and therefore the smoothness of the resulting estimates. The parameters π_j, ξ_{ji}, and $\boldsymbol{\theta}_j$ can be estimated from the data by the EM algorithm [4].

4 Self-Organizing Maps in the Learning Metric

As discussed in Section 2, the SOM algorithm consists of iterative application of two steps: finding the closest model vector (1) and updating it and its neighbors on the map (2). These two steps should now be modified to use the new metric instead of the Euclidean metric. When searching for the closest model vector we use the local approximation (3), computed around the data sample $\mathbf{x}(t)$. It is implicitly assumed that the approximation is locally accurate enough to preserve the identity of the winner, which is reasonable since the winner and its competitors are usually close to the data sample. There may be occasional errors in the winner search, and therefore the ultimate test for the validity of the assumption is whether the algorithm works for real-world data. This is tested in Section 5.

The winning unit is the unit c, for which

$$d^2(\mathbf{x}, \mathbf{m}_c) \leq d^2(\mathbf{x}, \mathbf{m}_i) \tag{8}$$

for all i. The distance d is given in (7).

In the second step of the SOM algorithm the winner and its neighbors are updated into the direction where the distance $d^2(\mathbf{x}, \mathbf{m}_i)$ decreases most rapidly, and proportionally to the magnitude of the change. In the Euclidean metric, the update is given by the gradient of the (squared) distance. The steepest descent in the new metric, which is technically a Riemannian metric, is expressed in the original coordinates of the $\mathbb{X}$-space by the so-called natural gradient [2], which can be shown to be here

$$\mathbf{J}^{-1}(\mathbf{x}) \frac{\partial}{\partial \mathbf{m}_i} d^2(\mathbf{x}, \mathbf{m}_i) = 2(\mathbf{m}_i - \mathbf{x}) \; . \tag{9}$$

Assuming that $\mathbf{x}$ and $\mathbf{m}_i$ are close to each other, (9) coincides with the direction of the shortest path from $\mathbf{x}$ to $\mathbf{m}_i$.

In conclusion, the update rule in the new metric is the same as in the Euclidean SOM, (2). The difference lies in the definition of the winner, (8), where the distance measure is defined by (7). Note that the Fisher information matrix need not be formed explicitly in computing the distance; the complexity of the distance computation is of the order of the dimensionality of the data times the number of possible values of c times the number of mixture components.

5 Data Analysis and Visualization

We applied the methods described in the previous sections to the task of visualizing and analyzing the bankruptcy risk of Finnish enterprises. We will first verify that a SOM in the new metric is more accurate in modeling the bankruptcy risk than a SOM in the Euclidean metric, and after that demonstrate how SOMs in the learning metric can be used for visualizing useful properties of the data. In this section we will call a SOM computed in the Euclidean metric SOM-E, and a SOM computed in the learning metric SOM-L. The primary data consisted of 6195 yearly financial statements given by Finnish small-to-medium-size companies. For each statement, 23 financial indicators were used. The auxiliary data was binary, indicating whether the company went bankrupt in three years or less.

The data was divided into roughly equal-sized training and test sets. PDF estimates were constructed for the training set with the MDA2 model having 10 mixture components. SOMs of the size of 20×10 units were then trained in the original Euclidean metric and learning metrics derived from the PDF estimates.

5.1 Accuracy of the SOM in the New Metric

To verify the goodness of a trained SOM quantitatively, we measured how well it represented the probability of bankruptcy in the input space. Each SOM unit can be used to represent and visualize the probability in one region of the input space (see Fig. 3). The accuracy of the representation can be measured as the likelihood of data within the region, that is, within the map unit. The overall goodness measure will be the average over map units.

If the SOM represents bankrupt risk well, then the measure will be close to the likelihood obtained directly from the PDF estimate that was used to form the metric. The likelihood of the PDF estimate will in general be an upper limit to the likelihood of the SOM, since the SOM projects the data to a finite number of units in order to visualize the data.

The likelihoods for the SOM-E and SOM-L are shown in Figure 2 as functions of σ, which controls the smoothness of the density estimates. The SOM-L clearly outperforms SOM-E. The difference becomes small for small σ; then the estimate and the resulting metric are probably uneven. Representation by the SOM-L therefore retains more of the probability density structure than representation by the SOM-E.

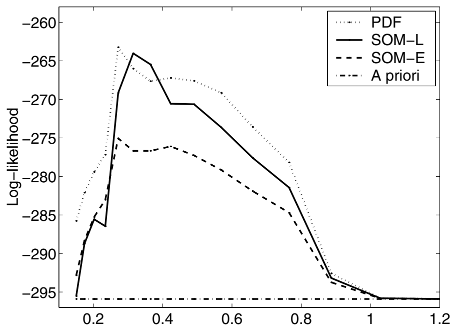


Fig. 2. The accuracy of the SOMs computed in the Euclidean metric (SOM-E) and in the learning metric (SOM-L) in representing the bankruptcy risk, measured by the likelihood of data projected to best-matching SOM units. The PDF is estimated by a Gaussian mixture with 10 mixture components. The curve marked by "PDF" provides an approximate upper limit: it is the likelihood of unprojected data. The curve marked by "a priori" is the lower limit of sensible results, given by the best constant estimates. The parameter σ governs the smoothness of the PDF estimates.

5.2 Visualizations for Exploratory Data Analysis

Each company has a location on the SOM display, and the similarity relationships of the companies can be visualized by displaying their names on the lattice. Several additional visualizations are possible. The first is to plot the value of the location-conditional bankruptcy probability on the SOM. This can be done in two ways: one can plot the probability given by the PDF estimate at the location of the map units. One can also directly plot the ratio of bankruptcy-prone and healthy companies at each map unit. The problem with the latter case is that the estimate is very noisy since the number of bankrupt companies is relatively small (77 in the training and 62 in the test set!).

Companies that went bankrupt are represented by a single area on the map (Fig. 3). Companies that did not go bankrupt form two separate areas, one on either side of the bankruptcy-bound companies. This could indicate that there are two separate subclasses of companies that avoid bankruptcy.

One can also visualize the values of each input feature (financial indicator) at the map units to inspect which kinds of companies are situated close to the bankruptcy zone. Three sample indicators are plotted in Figure 4. The indicators change in an intuitive manner: The profitability indicator and capital structure indicator decrease near bankruptcies. The liquidity indicator exhibits more complex behaviour, although it too decreases near bankruptcies. The SOM-L displays are more unimodal and well-ordered than the SOM-E displays: this may be because the SOM-E also represents data properties unimportant for bankruptcy risk.

A third possible kind of visualization is to plot for each map unit the contributions of the original variables in the metric or, equivalently, to the changes

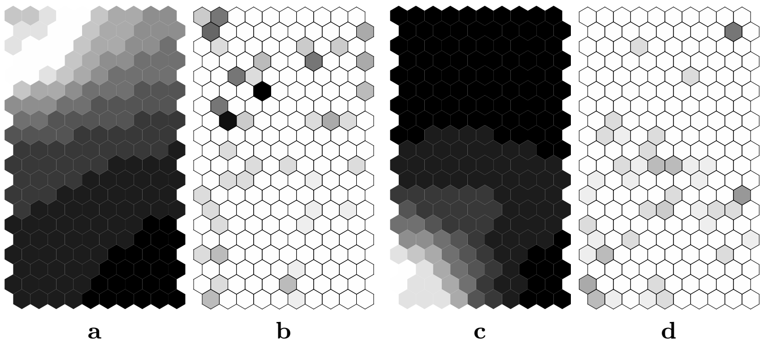


Fig. 3. The separation of bankruptcy-prone and healthy companies on the SOMs. **a** and **c**: The estimate of the probability of bankruptcy at each map unit in **a** SOM-L and **c** SOM-E. The estimate is the posterior density of the Gaussian mixture model at $\sigma = 0.3145$. The darkest shade denotes probability 0.002; the lightest denotes probability 0.119. The actual relative frequency of bankruptcies in the test set for each map unit is shown in **b** for SOM-L and in **d** for SOM-E. White: no bankruptcies, black: two thirds of all companies have gone bankrupt.

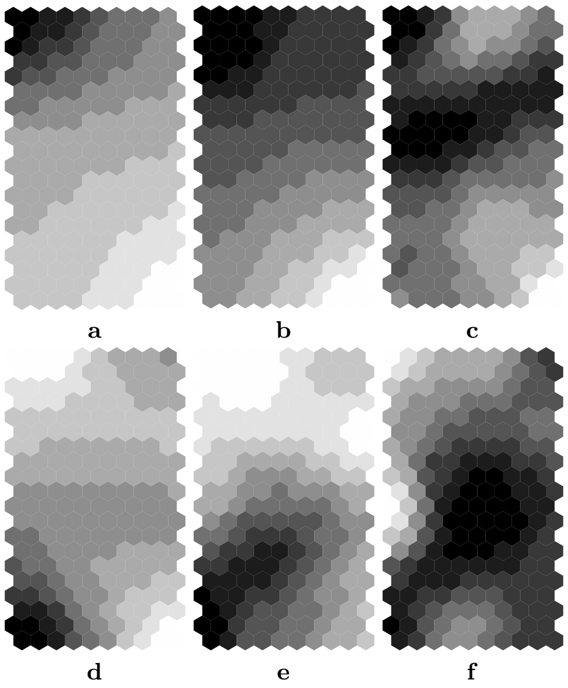


Fig. 4. The values of three financial indicators on (**a**-**c**) SOM-L and (**d**-**f**) SOM-E. An index of (**a** and **d**) profitability; (**b** and **e**) capital structure; (**c** and **f**) liquidity. The value of the smoothness parameter was $\sigma = 0.3145$.

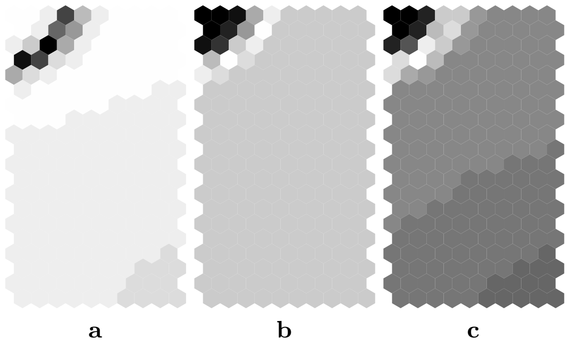


Fig. 5. The relative importances $r_l^2(\mathbf{x})$ of the same indicators as in Fig. 4 to the change in bankruptcy risk, shown as gray levels on the map display. An index of (**a**: scale from 0.007 to 0.080) profitability; (**b**: scale from 0.0002 to 0.215) capital structure; (**c**: scale from 0.001 to 0.013) liquidity.

in the bankruptcy probability. The contribution of variable l to the metric at $\mathbf{x}$ can be computed as

$$r_l(\mathbf{x}) = \sqrt{\frac{\mathbf{e}_l^T \mathbf{J}(\mathbf{x}) \mathbf{e}_l}{\sum_m \mathbf{e}_m^T \mathbf{J}(\mathbf{x}) \mathbf{e}_m}}, \tag{10}$$

which is the relative amount of scaling in the direction of the axis corresponding to the lth variable. Here $\mathbf{e}_l$ denotes the unit vector having the direction of the lth axis. If changing an input variable greatly increases the distance to the original location in the learning metric, then the input feature is important for bankruptcy risk near that location. For the original Euclidean metric, the relative importances are of course equal everywhere.

The three indicators shown in Figure 5 exhibit different behaviour near the bankruptcy zone. The importance of the profitability indicator decreases toward the bankruptcy zone, while the importance of the liquidity and capital structure indicators increase. The profitability indicator behaves in a similar fashion on both sides of the bankruptcy zone, but the importance of the capital structure and liquidity indicators remains small on one side. The presence of such nonlinear effects suggests that the use of nonlinear metric transformations is justified.

Other visualizations besides the ones shown here are also possible. One can visualize the overall amount of scaling at map units, which shows how stable those locations are. When the overall scaling is large, even small changes in the financial statements can have a large effect on bankruptcy risk. One can also plot the entire metric transformation matrix at various map units, in order to further visualize connections between the importance of the original input features.

If time series data is available, one can plot trajectories of data points on the map, and see how the financial statements change for different types of companies (cf. [15]). In the learning metric, it is also possible to plot trajectories of how the relative importance of different features changes over time.

6 Discussion

We have introduced a method for extracting interesting variation from data by learning a metric into the data space. The changes in the distribution of an auxiliary discrete variable, occurring in pairs with the primary data, are assumed to indicate interestingness. In addition, we have described how Self-Organizing Maps can be computed in the new metric to visualize the extracted features. In the case study, the method produced more accurate descriptions of the bankruptcy probability of companies than a SOM computed in the Euclidean metric. All the visualizations were smooth, and companies with a high bankruptcy risk became clearly separated from the rest. Thus the new metric was regular enough to retain the organization and visualization capabilities of the SOM while highlighting those aspects of the data that were relevant for the bankruptcy risk.

According to our knowledge the principle is new. Works with aspects from our approach exist, however. Jaakkola and Haussler [10] induced a distance measure into a discrete input space using a generative probability model. The crucial differences are that they do not use external information, and that they do not constrain the metric to preserve the topology. Hoffman [9] has also generated a metric based on a probability model.

More generally, our work is related to nonlinear projection (dimensionality-reducing) mappings. Various cost functions are used by these, including the mutual information [5,22]. Unlike in a standard separate feature extraction stage, however, the change of the metric in our method defines a manifold which cannot in general be projected to a Euclidean space of the same or lower dimensionality. Therefore, no dimensionality-preserving or dimensionality-reducing mapping with the same local properties exists which means that the change of the metric is a more general operation than feature selection by a dimensionality-preserving (or dimensionality-reducing) nonlinear mapping. Moreover, defining the metric in the original data space makes the results directly interpretable in terms of the original data variables.

Acknowledgments. The authors would like to thank Finnvera Ltd. and particularly Pentti Bergius for the data set, Kimmo Kiviluoto for his help regarding its interpretation, and the Academy of Finland for financial support.

References

1. Amari, S.-I.: Differential-Geometrical Methods in Statistics. Springer, New York (1990)
2. Amari, S.-I.: Natural Gradient Works Efficiently in Learning. In: Neural Computation **10** (1998) 251–276
3. Card, S. K., Mackinlay, J. D., Shneiderman, B. (eds.): Readings in Information Visualization. Using Vision to Think, Morgan Kaufmann, San Francisco, CA (1999)

4. Dempster, A. P., Laird, N. M., Rubin, D. B.: Maximum Likelihood from Incomplete Data via the EM Algorithm. In: Journal of the Royal Statistical Society, Series B **39** (1977) 1–38
5. Fisher, J. W. III, Principe, J.: A Methodology for Information Theoretic Feature Extraction. In: Proc. IJCNN'98, International Joint Conference on Neural Networks, Vol. 3. IEEE Service Center, Piscataway, NJ (1998) 1712–1716
6. Hastie, T., Tibshirani, R., Buja, A.: Flexible Discriminant and Mixture Models. In: Kay, J., Titterington, D. (eds.): Proc. Conf. on Neural Networks and Statistics. Oxford University Press (1995)
7. Hastie, T., Tibshirani, R.: Discriminant Analysis by Gaussian Mixtures. JRSSB (1996)
8. Holmström, L., Koistinen, P., Laaksonen, J., Oja, E.: Neural and Statistical Classifiers—Taxonomy and Two Case Studies. In: IEEE Transactions on Neural Networks **8** (1997) 5–17
9. Hofmann, T.: Learning the Similarity of Documents: an Information-Geometric Approach to Document Retrieval and Categorization. In: Solla, S. A., Leen, T. K., Müller, K.-R. (eds.): Advances in Neural Information Processing Systems 12. MIT Press, Cambridge, MA (2000) 914–920
10. Jaakkola, T. S., Haussler, D.: Exploiting Generative Models in Discriminative Classifiers. In: Kearns, Michael S., Solla, Sara A., Cohn, David A. (eds.): Advances in Neural Information Processing Systems 11. Morgan Kauffmann Publishers, San Mateo, CA (1999) 487–493
11. Kaski, S., Kangas, J., and Kohonen, T.: Bibliography of Self-Organizing Map (SOM) Papers: 1981–1997. Neural Computing Surveys **1** (1998) 1–176.
12. Kass, R. E., Vos, P. W.: Geometrical Foundations of Asymptotic Inference. Wiley, New York (1997)
13. Keim, D. A., Kriegel, H.-P.: Visualization techniques for mining large databases: A comparison. IEEE Transactions on Knowledge and Data Engineering **8** (1996) 923–938
14. Kiviluoto, K.: Predicting Bankruptcies with the Self-Organizing Map. Neurocomputing **21** (1998) 191–201
15. Kiviluoto, K., Bergius, P.: Exploring Corporate Bankruptcy with Two-Level Self-Organizing Maps. Decision technologies for computational management science. In: Proceedings of Fifth International Conference on Computational Finance. Kluwer Academic Publishers, Boston (1998) 373–380
16. Kohonen, T.: Self-organized Formation of Topologically Correct Feature Maps. In: Biological Cybernetics **43** (1982) 59–69
17. Kohonen, T.: Self-Organizing Maps. Springer, Berlin (1995; second, extended edition 1997)
18. Kullback, S.: Information Theory and Statistics. Wiley, New York (1959)
19. Murray, M. K., Rice, J. W.: Differential Geometry and Statistics. Chapman & Hall, London (1993)
20. Rao, C. R.: Information and the Accuracy Attainable in the Estimation of Statistical Parameters. Bull. Calcutta Math. Soc. **37** (1945) 81–91
21. Ripley, B. D.: Pattern Recognition and Neural Networks. Cambridge University Press, Cambridge, UK (1996)
22. Torkkola, K., Campbell, W. M.: Mutual Information in Learning Feature Transformations. In: Proc. ICML'2000, The Seventeenth International Conference on Machine Learning (2000)

Towards a Novel OLAP Interface for Distributed Data Warehouses

Ayman Ammoura, Osmar Zaïane, and Randy Goebel

The University of Alberta, Department of Computing Science, Alberta Canada
{ayman, zaiane, goebel}@cs.ualberta.ca

Abstract. We present a framework for visualizing remote distributed data sources using a multi-user immersive virtual reality environment. DIVE-ON is a system prototype that consolidates distributed data sources into a multidimensional data model, transports user-specified views to a 3D immersive display, and presents various data attributes and mining results as virtual objects in true 3D interactive virtual reality. In this environment, the user navigates through data by walking or flying, and interacts with its objects simply by "reaching out" for them. To support large sets of data while maintaining an interactive frame rate we propose the VOLAP-tree. This data structure is well suited for indexing both levels of abstraction and decomposition of the virtual world. The DIVE-ON architecture emphasizes the development of two main independent units: the visualization application, and the centralized virtual data warehouse. Unlike traditional desktop decision support systems, virtual reality enables DIVE-ON to exploit natural human sensorimotor and spatial pattern recognition skills to gain insight into the significance of data.

1 Introduction

The recent rapid development of data mining is a response to the fact that technology enables data collection, classification and storage at a rate far exceeding that with which we can analyze it [10]. To better support the operations usually associated with data analysis and mining, researchers have developed the concept of a data warehouse [11] to model voluminous data in a way that promotes the transformation of information into knowledge. Since vision is by far the human's most predominant sense, many researchers have targeted visualization as the means by which data is presented for analysis [3,7,15,16]. Our proposed system **DIVE-ON** (Datamining in an Immersed Virtual Environment Over a Network) takes visualization a step further by leveraging the human natural skills within an environment that simulates natural settings. Using the sensorimotor skills gained at childhood, one maneuvers through the natural world and acquires spatial knowledge almost unconsciously. To support such natural skills, we have constructed an Immersed Virtual Environment (**IVE**) that uses motion-trackers to acquire movement data, and then simulate the kinesthetic feedback through image transformation. This provides the user with a correlation between orientation and movement, to support a navigation interface that is transparent and highly capable in examining spatial correlations [3,15]. DIVE-ON combines advances in virtual reality (VR), computer graphics, data mining, and distributed data warehousing into one flexible system that can be used effectively with little or no training.

DIVE-ON constructs a virtual data warehouse from a set of distributed DBMS systems. Information needed during a visualization session is communicated between the visualization module and the virtual data warehouse as XML documents using CORBA or SOAP technologies. The data warehouse uses a global schema that describes the location of the information needed for building an N-dimensional data cube or any of its subsequently derived subsets. Once the immersed user specifies a particular view, the warehouse queries the individual sources, assembles the resultant cuboid as an XML document, and forwards it for visualization. Here we also present the **VOLAP-tree** (Visual OLAP tree), a special data structure designed to address the demands for real-time rendering and interactive OLAP operations through the recursive spatial decomposition of the materialized "OLAP regions."

Abstractly, DIVE-ON consists of three task-specific subsystems. Figure 1 shows the various layers comprising the complete system, from the data sources to the visualization environment. The first subsystem is the Virtual Data Warehouse (**VDW**) (see Figure 7), which is responsible for creating and managing the data warehouse over the distributed data sources. The second subsystem is the Visualization Control Unit (**VCU**), which is responsible for the creation and the management of the immersed virtual environment (IVE) to insure that the "reality" in virtual reality is not compromised (Figure 1 (3), details in Figure 5).

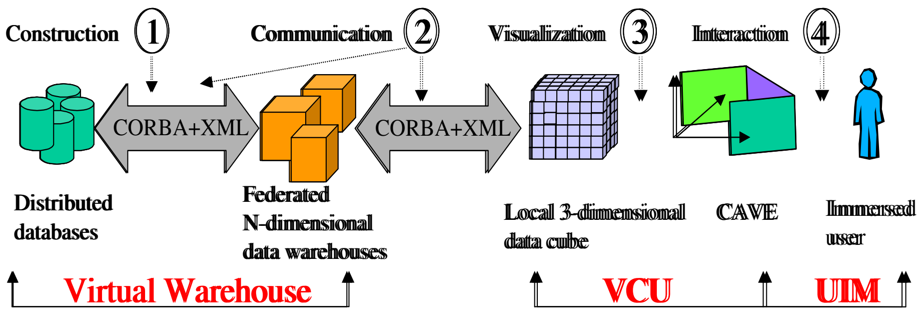


Fig. 1. The three components of the DIVE-ON system

The User Interface Manager (**UIM**) (Figure 1 (4), details in Figure 4) handles the direct application-control interaction as well as the automatic interaction that provides the kinesthetic feedback for navigation and aggregate manipulation. Inter and intra subsystem data exchange is provided by a set of specialized interfaces which implement specific protocols to guarantee extendibility and subsystem independence. This communication takes the form of client and server applications, using both Common Object Request Broker Architecture (CORBA) over TCP/IP and Simple Object Access Protocol (SOAP) over HTTP.

The rest of this paper is organized as follows. Our immersive display technology is presented, loosely corresponding to a CAVE. We include our motivation for using this environment, and virtual reality in general. We then present the software architecture of the Virtual Data Warehouse(**VDW**), and explain how the XML-based (XMDQL) queries are created and distributed amongst the various data sources.

2 Working in a CAVE

While information gathering and data warehouse management can be done from any location, the actual visualization experience takes advantage of the state-of-the-art virtual reality environment that is formally known as the **CAVE©** theater. CAVE is a recursive acronym (Cave Automatic Virtual Environment) [5], and refers to a visualization environment that utilizes tracking devices along with, up to six, large display screens. Our version places the user within three (9.5 X 9.5) feet walls (Figure2). Each of these walls is back-projected with a high-resolution projector that delivers the rendered graphics at 120 frames per second (Figure3). To simulate the way we perceive depth, the frame rate is divided into a left-eye channel and a right-eye channel (60 frames per second each). These two channels are synchronized with light weight shutter glasses that the user wears to create what is known as *stereoscopic* graphics.

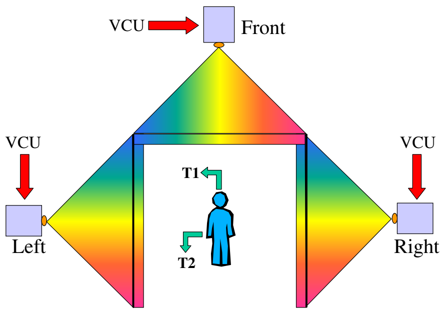


Fig. 2. A CAVE user within the three back-projected walls. T1, T2: The head and hand-held tracker data stream respectively (Real-time)

Using this type of environment for visualization over a desktop is justified by two psychophysical experiences; **immersion** and **presence** [3]. Regardless of how realistic the desktop graphics appear, the user is merely “looking at” a computer-determined point of view [15]. But within the walls of the CAVE the user is presented with an *egocentric* frame of reference which effectively immerses the user into VR. Standing at the center of the CAVE, the available field of view is a user-centric 270 degree angle. Users can span this view just as they would in a natural setting, by simply turning their heads [5]. Presence is the sense of "being there" and is enhanced by simulating the *kinesthetic* feedback gained while walking through a scene [11]. The user's head location and orientation within the three walls are monitored via a light-weight head tracker. The tracking information is used to update the views using the magnitude and direction of the user's most recent motion vector. Presence is also enhanced by the use of a hand-held wand that is used to perform OLAP operations, probe the data objects, control the environment, and navigate the IVE.

Immersion and presence, when enhanced by peripheral vision and depth perception, are important factors that help improve situational awareness, context, spatial judgment, and navigation and locomotion [15]. As argued by Pauline Baker [3], these factors makes navigation within a 3D model world *practically natural*, and dramatically easier than trying to maneuver around three dimensions using 2D-based desktop controls. Since explorative visualization should be thought of as a task driven and not a data driven process, the next section illustrates how our virtual objects are created in light of what we seek to accomplish.

3 Spatially Encoding Data as Visual Cues

DIVE-ON creates a visualization environment on a conceptual level and, unlike most iconographic data visualization systems, DIVE-ON is not primarily concerned with quantitative measures. For example, the Immersed Virtual Environment (**IVE**) is not designed to tell the user that the total sale of a branch was X dollars; rather it is designed to convey the significance of this amount with respect to its context. Once an "interesting" locality has been identified, the user is capable of extracting the original data lineage.

The primary abstraction of DIVE-ON is based on graphical rendering of data cubes. Selected data are extracted from the VDW (Sec. 4.4) after which relevant attributes are encoded in graphical objects and then rendered in the virtual world. The VCU interprets the three-dimensional cube it receives from the VDW as a three variable function (Sec. 4.2). Each of the three data dimensions is associated with one of the three physical dimensions, namely X, Y, and Z. Since each entry in the data cube is a structure containing two measures M_1 and M_2, the VCU simply plots the two functions M_1(x, y, z) and M_2(x, y, z) in $\Re^3$.

We recognize that there are may alternatives for encoding the data cube measures as graphical objects. Our current prototype uses cube or sphere size and colour to provide the user with visual cues on measure contrasts. For example, if we are focused on the theme "dollars sold" (Figure 3), we assume the OLAP user is not primarily interested in the details that in year t the total sale of product p at store s was $100,000.00. Instead, the VCU provides a context by associating these measures with visual cues that are bound to the virtual objects. In this case, the first cue we use is *size*, which is associated with the measure M_1 (dollars sold). After normalization, $M_1(x_t, y_p, z_s)$ is used to render a cube (or a sphere) of appropriate size, centered at position (x_t, y_p, z_s), for some t, p, and s within the data range, as shown in Figure 3. A VR user "walking" among these virtual objects becomes almost instantly aware of the relative significance of each value, without the need for specific numeric data.

Our prototype uses an object's *colour* as a second visual cue, by normalizing a measure M_2, and mapping to a discrete 8-colour palette. For example, we can encode any abstract data mining "interestingness" measure from "red" to "blue." For example, at the lowest level of aggregation (high granularity), colour can represent the deviation from the mean along one of the dimensions. This is particularly useful for market fluctuation analysis. Similarly, if the user is viewing highly summarized data, colour can be a very effective way to locate anomalies at a lower level. For example, the M_2 value for a month object can represent the maximum M_2 of any of the days it aggregates. In Figure 3, each virtual object represents the total revenue for a given year. The colour "red" indicates that one particular month deviates significantly from the rest of the year. We expect the OLAP analyst will *reach* in virtual reality and "select" that object, in order to understand the deviation, and inquire about the exact figures for that year. Similarly, the user may be interested in understanding the stability which dominates a product category (a "blue" object).

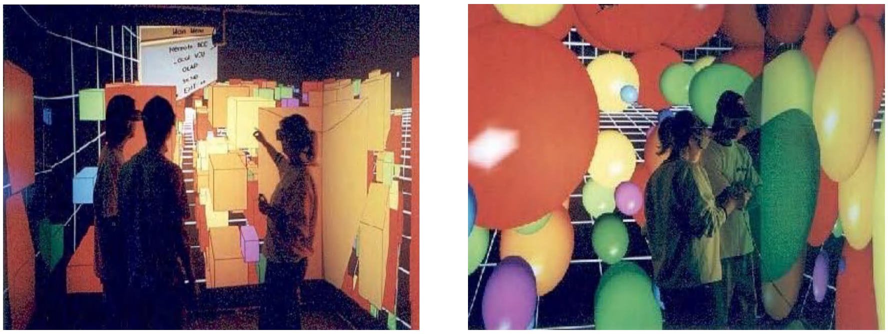

(a) (b)

Fig. 3. A team of immersed users discussing the "dollars sold" data cube. (a) Using cubic objects (b) A user pointing the direction of flight within 3D-lit spheres

Figure 3 presents the IVE created by rendering cubes that employ the visual cues described above. In this case, the X-axis (left to right) represents the "product" dimension. The axis pointing in the direction perpendicular to the picture is the "time" dimension Y, while Z represents "location." (The floating 3D interaction menu is also visible.)

Our brief discussion presented cubes as the basic VR geometry, but DIVE-ON can also use spheres in the same way. While spheres can encode the same information as cubes with less occlusion [1], rendering spheres is computationally much more expensive. To create a 3D sphere the system must compute light sources, normal vector calculations, material specification, and shade rendering. None of these calculations are required for cubes, since the polygon rendering is typically done by hardware.

4 System Design and Architecture

To flexibly support various VR devices, tools, and environments it was necessary to separate the creation from the application of the virtual world. Each of the three main DIVE-ON components is designed within a wrapper that defines the mean for information exchange. In this section the main components that make up the system are presented along with the the task specific design and implementation issues.

4.1 UIM: The User Interface Manager

The User Interface Manager (UIM) is the subsystem that is responsible for receiving, filtering, and channeling all available input streams. Input examples include the location and orientation of the tracking devices, and the button-status on the hand held tracker (Figure 4). This information must be updated at a sufficient rate to provide a natural smooth interaction with the environment. To provide the sense of immersion and presence, the VCU reads the head tracker motion data collected by the UIM (T1 in Figure 2), then transforms the stereo graphics to simulate that motion in a physical world. For example, if the user walks forward, the appropriate image is shifted backwards to create the illusion of "walking" through the data. The data stream emitted from the user's hand (T2 in Figure 2), is used to track the position of the 3D menu in the virtual world. These so-called "floating menus" represent the user's hand to six degrees of freedom (6-DOF) [9].

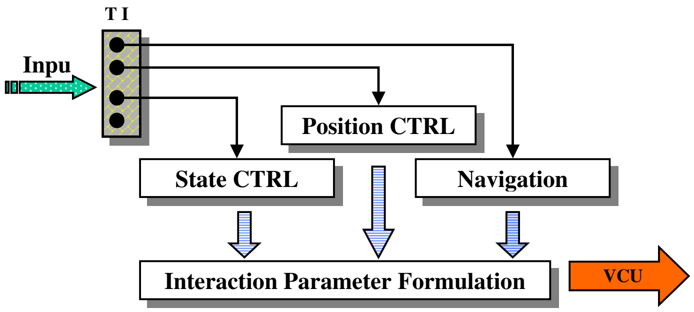


Fig. 4. User Interface Manager. The Tracker Interface (TI) receives the real-time tracker input stream and channels it according to type. The set of interaction parameters is then fed to the VCU

Using the floating menu system, the user is able to perform all application control commands, including various OLAP operations, in a natural manner. For instance, to perform a "roll-up" operation the user activated the menu system, selects "roll-up" and then, using the hand held tracker, points to the dimension to be rolled-up. The operations of "slice," "dice," and "drill-down" are implemented similarly (Figure 3). In a typical session, a client (VCU) first establishes connection to the VDW and, using warehouse queries, the user can then inquire about the number, size and attributes of each data dimensions available.

Viewpoint manipulation is implemented by navigation control. In views that involve dimensions with large domains, the user may request the activation of *flight mode*. In this mode, the user travels through the data by simply pointing in the appropriate direction. Flight speed is determined by how far the arm is extended away from the body. Finally, DIVE-ON also provides the user with the ability to inquire about the original data that is represented by a given virtual object. The hand-held tracker default mode is 3D *virtual pointer*. If an "interesting" data aggregate is encountered, the user can point and pop up an object-fixed window containing the particular aggregate lineage.

4.2 VCU: The Visualization Control Unit

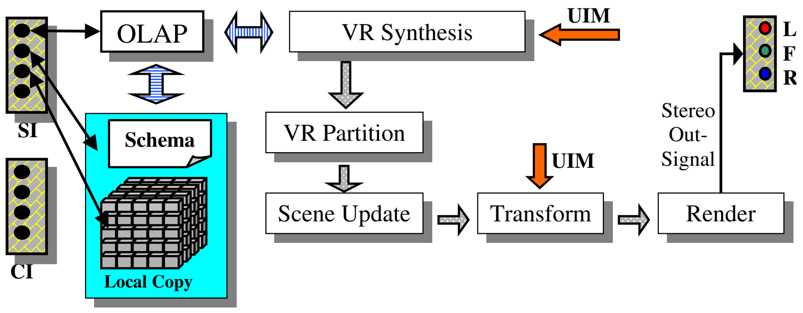


Fig. 5. The VCU Architecture. **SI** and **CI** are the SOAP and CORBA client Interfaces respectively. Output is channeled to Left, Front, and Right projection stereo signals

The VCU is the module responsible for generating and managing the IVE. This makes data visualization and exploration independent, so the specifics of the VDW should be of no concern to the VCU developer and vice versa. To implement this abstract view, the VCU and VDW are each constructed within a wrapper that isolates the only method of relaying messages between the two subsystems. The messages use a simple communication protocol which effectively hides the implementation details and allows the VCU and the DCC to be independent of one another. After the DCC completes the creation of the N-dimensional data cube it signals the VCU (via the DCC-Shell). Since we are generating a 3D virtual world, only three dimensions can be viewed at any given time (four dimensions is possible by enabling animation using a function we call“animating the data through time” [1]). The three dimensions and associated measures selected by the user are extracted from the N-dimensional data cube, then a 3D cube is passed to the VCU for rendering. In light of the above discussion, what level of abstraction does that the 3D cube represent? If the cube received is already summarized, then for every “roll-up” that the user requests a new 3D cube must be requested. This imposes unnecessary strain on the network and degrades the system’s interactivity. For this reason, the VCU builds a working 3D copy that materializes different levels of abstraction into imbedded OLAP regions that are indexed by the VOLAP-tree (Section 4.3).

4.3 VOLAP-Tree: A Spatial Decomposition Structure

Data is obtained from the VDW along with the concept schema and the concept hierarchy that describe the associated aggregation method. To accommodate partial ordering, a concept hierarchy is presented by a simple tree structure with the root being the attribute “ALL” at level 0. The information is then used by the VCU to construct a *working cube* consisting of a set of OLAP regions that provide all possible views (differing granularities) of the user specified dimensions. Each region is a contiguous chunk that can be identified by two vectors. To illustrate using a simplified example, consider Figure 6 which represents a 2D slice of the working cube. The view corresponding to region B (Province/Item) is identifiable by the vectors (X_2, Y_0) and (X_3, Y_1).

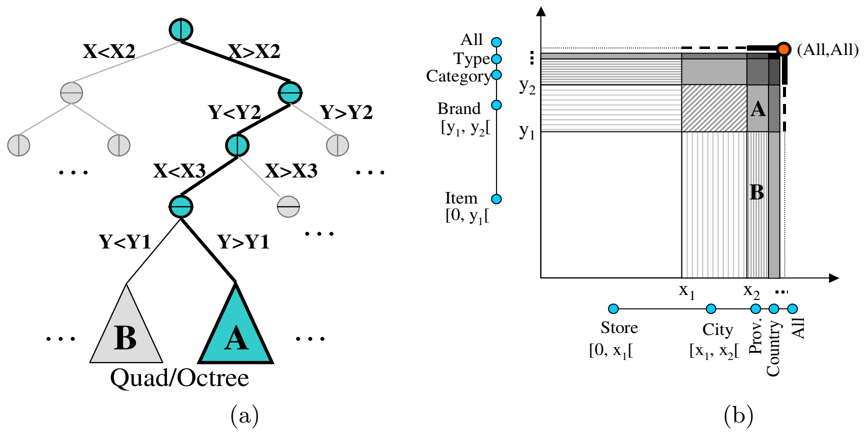


Fig. 6. (a) The VOLAP-Tree. (b) Materialized working cube within the VCU

The simplest mapping between the user’s location in virtual space and the location in the data space is to correlate discretized points in VR with the index values of the working cube. One can imagine that this slice as the "floor" of the CAVE and when the user maneuvers through VR, the center of the rendered scene is updated to reflect the cell that the user is closest to. This design is closely related to the used metaphor of "walking through the data cube." The efficiency of this design stems from the fact that performing OLAP operation is simply equivalent to "transporting" the user to a different region within the working cube. As an example, consider the two associated concept hierarchies that are shown next the axis that maps them in VR (Figure 6 b). Domain attributes of the dimension "location" at the lowest level, "store," are assigned the index values between $[0, X_1[$ along the X axis. Similarly, the attributes defining "categories" in the "product" dimension are assigned the range $[Y_2,\ Y_3[$. Assuming that the current view is (Province/Item), region B, specifying a "roll-up" operation on the dimension "product" is equivalent to transporting the user into region A.

The VOLAP-tree is a hybrid that includes a 3D-tree (KD-tree) and a set of Octrees that is designed to quickly transport the user into different OLAP regions while maintaining an acceptable frame rate regardless of data size. Figure 6 (a) illustrates a 2D version of the tree. A 3D-Tree is implemented as an upper layer to index the OLAP regions within the working cube . The root of the VOLAP-tree is the root for the 3D-tree. At the leaf level, each 3D-tree leaf contains a pointer to the second layer, which is an Octree that recursively partitions that particular OLAP region into octants. Recursion continues until all octants at the leaf level do not contain more than a given number of data points. Within the VCU, the "VR Partition" module (Figure 5) uses the the user's location (UIM input) to determine the appropriate set of octree nodes to use for rendering. As the user's position changes through VR, so does the set of rendered octants. When the user performs an OLAP operation the VOLAP-tree is traversed and the new Octree root is located for rendering.

4.4 VDW: The Virtual Data Warehouse

The Virtual Data Warehouse (VDW) is abstract centralized data warehouse comprised of a set of distributed data sources and a shell (DCC-Shell) which is responsible for modeling and querying these sources (Figure 7). The DCC-Shell is maintains a pool of meta-data (*cube schema*) that represents a global multidimensional model of all dimensions and measures available from the distributed sources. When the VDW is initiated, the cube schema is constructed and copied to all data sources. To insure query consistency, the DCC-shell updates all copies when a data source has been updated. A resource allocation table within the DCC-shell maintains the location, data organization, and preferred communication method for each source. The cube schema and the resources data are XML documents for easy maintenance, extendibility, and flexibility.

A client has three available query classes. First is the *Warehouse query*, which provides the client with basic VDW structure including the dimensions of a data cube, the measures available, and the main theme of a cube. The *Cube schema query* provides the meta-data of one specific data cube in the VDW. This meta-data includes a depiction of all available dimensions, measures, and

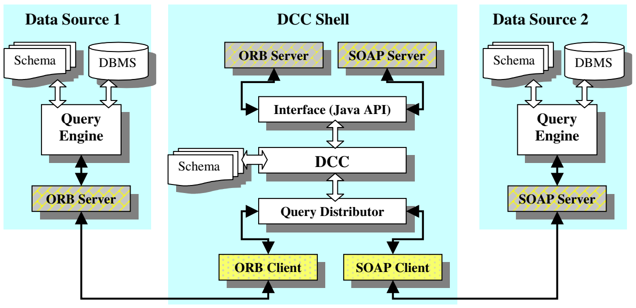


Fig. 7. The Virtual Data Warehouse (VDW) architecture

the concept hierarchy that further describes each dimension. Finally, the *Cube data query* is used to obtain an entire N-dimensional cube or any subset of it. This is particularly useful for applications such as the VCU, which handles only one 3D cube per visualization session.

4.5 XML Multidimensional Query Language (XMDQL)

Our query language choice is an XML-based query language, XMDQL, which we use to interact with the VDW in order to manage and access the available data. XMDQL allows the user to express multidimensional queries on the VDW. The concept of a special multidimensional query language was first proposed as an industry standard by Pilot software [14]. Their language MDSQL, however, does not take advantage of XML and its flexibility and interoperability in the context of federated data warehouses. In OLAP terminology, this type of query is equivalent to slicing and dicing the data cube. The result of an XMDQL query is a cell, a two-dimensional slice, or a multidimensional sub-cube.

DIVE-ON defines XMDQL as a query language that is formatted in XML to query the VDW; it also provides functionality similar to Microsoft's MDX (Multidimensional Expressions). To specify a cube, an XMDQL query must contain information about the four basic subjects: (1) The cube being queried, (2) dimensions projected in the result cube, (3) slices in each dimension and (4) some selection and filtering constraints. The basic form of the XMDQL is as follows:

```
<XMDQL>
   <SELECT>
     Project dimensions and slices
   </SELECT>
   <FROM>
       Which cube to query
   </FROM>
   <WHERE>
     Filtering constrains
   </WHERE>
</XMDQL>
```

4.6 Query Distribution and Execution

According to Figure 7, a VCU data query is first received by the DCC-Shell interface (through ORB/SOAP Server) and forwarded to the **Query Distributor** after the DCC has formed the appropriate XMDQL query. The Query Distributor analyzes the query, finds which data source contains the required data, and then distributes the query accordingly. Each Data Source executes the query separately, either by translating the XMDQL query into another OLAP query language and getting the result, or by directly accessing the original data source. Regardless of the execution method, each Data Source forms the results as a data cube that is returned to the DCC, which forms an N-dimensional data cube. For our visualization, the DCC then extracts the VCU-requested 3D data cube and sends it to the CAVE for rendering. To illustrate, the distribution information can be stored as following:

```
<Distribution dimension="Store">
  <Component path="N_America.USA"
             mart="DataSource1">USA sales data</Component>
  <Component path="N_America.Canada
             mart="DataSource1">Canada sales data</Component>
</Distribution>
```

5 Conclusion and Future Directions

We have presented a system prototype for visual data mining in an immersed virtual environment. Since the very early days of computing science with extremely limited technologies, scientists have been fascinated with virtual reality (VR). VR systems are capable of abstracting complex problems or scenarios by exploiting the human's natural skills including the visual system and spatial knowledge acquisition. The CAVE theater is a new technology that enables algorithms to interact with the human sensorimotor system. With DIVE-ON, we have focused this technology into a new direction, namely remote visual data mining.

So far we have exploited the human visual system to convey information pertaining to data. In the near future we also plan to experiment with *data sonification* techniques to add audible cues to the IVE. Since hearing is usually a background process, DIVE-ON will use the audible cues mainly to steer the user's foreground process, vision, into a direction that may need in-depth examination. Limiting the use of audible cues in this manner avoids the permeation of the IVE with sensory input that could lead to some undesired perceptual complexities. We also plan to investigate is the use of distortion views, also called fisheye or detail-in-context, in 3D graphics. Creating distortions in the 3D data cube, like creating a virtual magnetic field with a repelling force around interesting data items, can solve some of the occlusion problems by emphasizing relevant data and putting details in context. However, creating a fisheye effect on local detail in a virtual reality environment without compressing the remainder of the data is not trivial. In addition we plan to merge the interaction operations for distorted views in 3D with OLAP operations, such as aggregating local details, specializing and generalizing on a local detail, etc.

References

1. Ammoura, A., Zaïane, O. R., and Ji, Y., "Immersed Visual Data Mining: Walking the Walk," Proc. 18th British National Conference on Databases (BNCOD'01), Oxford, July2001.
2. Agarwal S., Agrawal R., Deshpande P., Gupta A., Naughton J. F., Ramakrishnan R. and Sarawagi S., "On the Computation of Multidimensional Aggregates," Proc. of VLDB Conference, 1996, pp 506-521.
3. Baker, M. P., "Human Factors in Virtual Environments for the Visual Analysis of Scientific Data," NCSA Publications: National Centre for Supercomputer Applications.
4. Chaudhuri, S., and Umeshwar, D., "An Overview of Data Warehousing and OLAP Technology," Proc. ACM SIGMOD Record, March, 1997.
5. DeFanti, T. A., Cruz-Neira, C., and Sandin, D. J., "Surround-Screen Projection-Based Virtual Reality: The Design and Implementation of the CAVE," Proceedings of ACM SIGGRAPH, 1993, http://www.evl.uic.edu/EVL/VR/systems.shtml.
6. Extensible Markup Language (XML): http://www.w3.org/XML/
7. Foley J. and Ribarsky, B., "Next-Generation Data Visualization Tools." In Scientific Visualization Advances and Challenges, chapter 7, pp 103-127. Academic Press/IEEE Computer Society Press, San Diego, CA, 1994.
8. Gary, J., Chaudhuri, S., Bosworth, A., Layman, A., Reichart, D., and Venkatrao, M., "Data Cube: A Relational Aggregation Operator Generalizing Group-by, Cross-Tab, and Sub-Totals," Proc. of the Twelfth IEEE International Conference on Data Engineering, February, 1996, pp 152-159.
9. Green, M. and Shaw, C. develop MR-Toolkit at the University of Alberta: http://www.cs.ualberta.ca/˜graphics/MRToolkit.html
10. Han J., and Kamber M., "Data Mining: Concepts and Techniques," Morgan Kaufmann Publishers, 2001.
11. Hand, C., "A Survey of 3D Interaction Techniques," Computer Graphics Forum, December, 1997, 16(5), pp 269-281.
12. Jaswal, V., "CAVEvis: Distributed Real-Time Visualization of Time-Varying Scalar and Vector Fields Using the CAVE Virtual Reality Theater," IEEE Visualization, 1997, pp 301-308.
13. Keim D. A., Kriegel H.-P.: VisDB: A System for Visualizing Large Databases , System Demonstration, Proc. ACM SIGMOD Int. Conf. on Management of Data, San Jose, CA, 1995.
14. Pilot Software: http://www.pilotsw.com/news/olap_white.htm
15. van Dam, A., Forsberg, A. S., Laidlaw, D. H., LaViola J. J., and Simpson, R. M., "Immersive VR for Scientific Visualization: A Progress Report,"Proc. IEEE Virtual Reality, March, 2000 (VR2000).
16. Ward, M. O., Keim D. A.: Screen Layout Methods for Multidimensional Visualization, Euro-American Workshop on Visualization of Information and Data, 1997.

Matchmaking for Structured Objects

Thomas Eiter[1], Daniel Veit[2], Jörg P. Müller[3], and Martin Schneider[3]

[1] TU Vienna, Institute of Information Systems, Knowledge Based Systems Group, Favoritenstrasse 11, A-1040 Vienna, Austria
eiter@kr.tuwien.ac.at
http://www.kr.tuwien.ac.at

[2] Universität Karlsruhe (TH), Information Management and Systems, Englerstrasse 14, D-76131 Karlsruhe, Germany
veit@iw.uni-karlsruhe.de
http://www.iw.uni-karlsruhe.de

[3] Siemens AG, Corporate Technology, Intelligent Autonomous Systems, Otto-Hahn-Ring 6, D-81739 Munich, Germany
joerg.mueller, martin.schneider@mchp.siemens.de
http://www.siemens.de

Abstract. A fundamental task in multi-agent systems is matchmaking, which is to retrieve and classify service descriptions of agents that (best) match a given service request. Several approaches to matchmaking have been proposed so far, which involve computation of distances between service offers and service requests that are both provided as aggregates of the same set of attributes which have atomic values. In this paper, we consider the problem of matchmaking in the setting where both service offers and requests are described in a richer language, which has complex types built from basic types using constructors such as sets, lists, or record aggregation. We investigate methods for computing distance values of complex objects, based on a generic combination of distance values of the object components, as well as domain-dependent distance functions. The methods have been implemented in GRAPPA, the Generic Request Architecture for Passive Provider Agents, which is a framework for developing open matchmaking facilities that can handle complex objects described in XML. Using GRAPPA, a large scale application has been built in the Human Resource Network project of the Office for Labor Exchange of the German government, in which job offerings have to be matched against a large database of unemployed persons and qualified candidates should be retrieved.

Keywords: Emerging trends; data warehouses; systems and applications.

1 Introduction

Today, distributed and heterogeneous information systems which are connected via open networks such as the Internet provide a huge, wide spread wealth of information. The vast amount of information so accessible has created a strong need for powerful methods and techniques that help in ranking the information retrieved in answer to a given query.

In multi-agent systems, this problem instantiates to the fundamental issue of classifying and ranking agents in a system by their service descriptions, given that a particular service is requested. For this task, special kinds of middle agents have been proposed, among them matchmaking and brokering agents (see [12,9,11]), following the mediator approach [17].

The key task in matchmaking is to compute the similarity of a given service request to the service description of a given agent. This is usually done by computing a function measuring the distance between the service request and the service description. Current approaches to matchmaking assume that these descriptions are in a flat format, which essentially is an aggregation of attributes over elementary domains. Distances are computed using well-known methods for computing the distance between values for a single attribute.

However, no methods for matchmaking of complex, structured service descriptions have been provided so far. Such methods are needed, though, for emerging applications that desire services descriptions in a data format which is structurally rich, and, moreover, obeys to some acknowledged standard (e.g., XML) such that multiple, application-independent use of this information is supported. In this paper, we address this issue and investigate methods for matchmaking of service descriptions which are provided as complex objects, built over possibly heterogeneous data types such as text, intervals, or time data, using forms of aggregation such as sets, lists, or records. We pursue a bottom up approach of combining distance values of components of complex objects into a single distance value, which may use generic combination functions as well as customized domain-dependent distance functions.

The main contributions of this paper can be summarized as follows:

- We provide methods for calculating the relevance of a service offer for a requested service, both given as structured complex objects, through distance functions for complex objects which combine distance values of their components. The latter may be complex objects as well, built using common forms of aggregations such as lists, sets, and records. Different from previous matchmaking approaches, ours is not based on a fixed scheme but works for *generic types* of service descriptions. Furthermore, service requests and offered service offers may be of different (yet fixed) type.
- We present the GRAPPA framework, which facilitates the development of matchmaking applications involving complex service and request descriptions. At the generic level, the schemes of the descriptions are stored as *XML document type definitions* (*XML-DTDs*). GRAPPA provides a number of predefined generic functions for combining distance values of description components, and furthermore certain domain-specific distance functions.
- We report on the Human Resource Network (HRNET), which is a large-scale application for employment relaying that has been implemented for the German Office for Labor Exchange on top of GRAPPA. In this application, hiring requests of employers have to be matched against the database of persons in Germany seeking a job (currently, about 3.9 millions), in order to single out best-qualified candidates. Experiments have shown that HRNET performs well, and a full-fledged system is planned for the future.

Our results, and in particular the GRAPPA framework, can be readily applied in building matchmaking facilities for agent systems. However, since the methods have

been developed at a generic level, they can be used for matching complex objects against a database of complex objects in general, and in engineering facilities for this task.
In this paper, we focus on the task of matchmaking per se. We pursue an approach in which the generic structure of offers and requests is given by offer and request class descriptions, respectively.

2 Matchmaking Facilities for Complex Objects

We contract here matchmaking to the problem of performing multidimensional distance computation on structured objects, which are composed bottom up from basic and complex values. We next discuss the components which a generic facility for performing this task should have, and after that in Section 2.2 the matchmaking process.

2.1 Components

As discussed in [12,2,9], a matchmaking facility needs certain components. In this spirit, we propose that a matchmaker for complex objects has the following four components: Data Types, Distance Functions, Service Scheme, and Service Repository. They have the following roles.
Data Types: This component includes a kernel set of basic types $\mathcal{B} = \{\tau_1, \ldots, \tau_n\}$ which are supported by the matchmaking facility. This kernel may be extended by customized types in applications. Complex types can be inductively constructed by applying one of the following constructors. Let τ and $\tau_1, \ldots, \tau_k$ be any already defined types: Set of τ, denoted $\{\ \tau\ \}$; Multiset of τ, denoted $\{\ \tau\ \}_m$; List of τ, denoted τ^+; Array of dimension n of τ, denoted $\tau[1:n]$; Record of $\tau_1, \ldots, \tau_k$, denoted $(\tau_1, \ldots, \tau_k)$.

Each basic type τ has an associated domain $D(\tau)$ of values. For a complex type τ, its domain of values $D(\tau)$ is defined by recursion to subtypes as usual, where sets, multisets, and lists are defined as finite aggregations. Note that different types may have overlapping or even the same sets of values; thus, subranges and synonym types may be defined.
Distance Functions: A *distance function* on a domain $D(\tau)$ is a map $d : D(\tau) \times D(\tau) \to \mathbb{R}_0^+$ which assigns each pair (x, y) of values for τ a unique nonnegative real number. It is desired that d enjoys certain properties, which guarantees a meaningful behavior. The following are some well-known axioms for distance functions:

(i) $d(x, y) = 0 \Longleftrightarrow x = y$ for all $x, y \in D(\tau)$. (Zero-Distance)

(ii) $d(x, y) = d(y, x)$ for all $x, y \in D(\tau)$. (Symmetry)

(iii) $d(x, z) \leq d(x, y) + d(y, z)$ for all $x, y, z \in D(\tau)$. (Triangle Inequality)

Any d which satisfies (i)–(iii) is a *metric distance function*. Its properties, in particular (iii), may be exploited for pruning the search space in matchmaking. However, not all meaningful distance functions in practice are metric. We postulate, though, that each distance function must satisfy (ii) and the if-direction of (i) (i.e., $d(x, x) = 0$ for all $x \in D(\tau)$).

The matchmaking facility must support for each type τ at least one distance function d. In particular, it must contain at least one *atomic distance function* for each basic type. Besides tailored distance functions, a complex type τ may have, as in GRAPPA, generic distance functions $d = f(d_1, \dots, d_m)$ which combine, by functions f, the distance values at the top-level components $\tau_1, \dots, \tau_m$ of τ, computed using respective distance functions $d_1, \dots, d_m$, into a single distance value. For example, in case of a record $\tau = (\tau_1, \tau_2)$ the function f may be the average of the distances at the components τ_1 and τ_2 (see [14] for further discussion).

Service Description Scheme (SDS): This component contains generic descriptions of the service offers and requests, given as types, that the matchmaking facility can handle. Particular service descriptions are instances of these generic descriptions (i.e., complex values of the types). The **SDS** consists of the following three parts:

1. OSDS: A scheme (type) for the definition of a service offer.
2. RSDS: A scheme (type) for the definition of a service request.
3. MAP: A mapping which assigns each component $\mathtt{R_i}$ of the scheme RSDS = $(\mathtt{R_1}, \dots, \mathtt{R_m})$ a function

$$\mathtt{MAP}(\mathtt{R_i}) : D(\mathtt{OSDS}) \to D(\mathtt{R_i}),$$

such that $\mathtt{MAP}(\mathtt{R_i}) = f_{\mathtt{R_i}}(\mathtt{S_1}, \dots, \mathtt{S_k})$, where $f_{\mathtt{R_i}}$ is a function on functions and $\mathtt{S_1}, \dots, \mathtt{S_k}$ are subschemes of OSDS, viewed as deconstructor functions on the instances of OSDS. Informally, $\mathtt{MAP}(\mathtt{R_i})(o)$ constructs for the service request component $\mathtt{R_i}$ a value, given any service offer o. We thus can construct a request object $\mathtt{MAP}(o) = (\mathtt{MAP}(\mathtt{R_1})(o), \dots, \mathtt{MAP}(\mathtt{R_m})(o))$ from o, which can be used for computing the distance between o and a given request object r. Any non-record type RSDS is viewed as record type (RSDS), and MAP defined for it this way.

Notice that in current matchmaking systems, OSDS and RSDS coincide, and MAP is identity, i.e., $\mathtt{MAP}(\mathtt{R_i}) = \mathtt{R_i}$. There, MAP is identity for several components (e.g., MAP(Workstatus) = Workstatus, where we use Workstatus to name the component of this type), while it assigns to $\mathtt{Profession}^+$ the list obtained by concatenating DesiredJob and $\mathtt{Job}^+$, i.e., $\mathtt{MAP}(\mathtt{Profession}^+) = \mathtt{DesiredJob@Job}^+$, where "@" denotes concatenation of lists (as previously, components are named here by their types).

Special cases are that RSDS is a subscheme of OSDS and vice versa. Here MAP is straightforward: in the former case, it projects out components of OSDS, while in the latter, $f_{\mathtt{R_i}}(\mathtt{S_1}, \dots, \mathtt{S_k})$ may add missing attributes to a service offer and assign dummy values to them. In practice, $f_{\mathtt{R_i}}$ may perform various complex operations such as merging lists, taking the union of sets, combining values into a complex value (e.g., assemble dates) etc.

Service Repository: The matchmaking facility maintains an up-to-date repository of service descriptions for all services offers which are advertised to it.

2.2 Matchmaking Process

When the matchmaker receives a *query*, which consists of an instance r of RSDS, and an (optional) query requirement (best match, k-nearest neighbors, etc), then it computes the answer to the query and sends the result back to the querying requester agent. Basically, the matchmaker must compute the distance between r and each service offer o in the service repository, and then select those o which qualify for the answer. The distance between r and o is measured by $d(r, \texttt{MAP}(o))$, where d is the distance function for `RSDS` and $\texttt{MAP}(o)$ is the conversion of o into the request object.

This process can be implemented in many ways. For details concerning this issue see [14,5,15].

2.3 The Grappa Matchmaking Framework

Grappa, the **G**eneric **R**equest **A**rchitecture for **P**assive **P**rovider **A**gents, is a generic framework that instantiates the general matchmaking facility for complex objects described in the previous section. It is designed for computing k-nearest-neighbor matchings of multidimensional requests against multidimensional offers.
Generic algorithms to incorporate Data Types, Distance Functions and Service Description Schemes are implemented. The main Data Types considered are Numbers, Intervals, Time and Time Intervals as well as Free Text. For the latter one distance functions from the Information Retrieval domain (see [10]) are implemented. For further details see [14, 5].
As complex distance functions Minimum Link Distance (see [4]), Hausdorff Distance as well as a Weighted Average Distance are incorporated.

Both the `OSDS` and the `RSDS` are mapped to XML-DTDs. Service offers and requests, respectively, are instantiated XML documents. An example is shown in Figure 1. A requester can query the **ServiceRepository**, which contains XML documents instantiating `OSDS`, by sending an XML document which instantiates the `RSDS` to the Grappa matchmaker.

3 Application

Because of its genericity, our approach and the Grappa framework is not restricted to agent systems and can be applied in different domains.

This is exemplified by two projects in which Grappa has been applied so far: The Human Resource Network (HrNet), a large-scale application for the mediation of jobs described in Section 3.1, and the Cooperation Market (CoMa) of the Siemens AG. In the following we will only report on the HrNet project.

3.1 HrNet for the German Office for Labor Exchange

The Human Resource Network (HrNet) is an application of Grappa for matching open jobs in companies, which are defined by an employer, to profiles of job applicants (i.e., unemployed persons), stored in various data bases. The current version of HrNet is a prototype system that has been developed for the Office for Labor Exchange of the German government, and demonstrates the feasibility of a partially automated approach to employment relaying. Based on its success, a full-fledged system is planned for the near future. Note that it promises a high return of investment: reducing the relaying time

Table 1. RSDS of HRNET as XML-DTD and an instance (computer scientist)

XML-DTD for RSDS:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT RSDS (Profession*, Experience, CarRequired,
                Location, Requirements, WorkStatus,
                WorkMode, WorkType, Salary)>
<!ELEMENT Experience (ExperienceLevel, Profession*)>
<!ELEMENT Location (ZipCode, Town, Country)>
<!ELEMENT Requirements (GeneralRequirement,
          SpecialRequirement, Language)>
<!ELEMENT GeneralRequirement (Description, Level)>
<!ELEMENT SpecialRequirements (Description, Level)>
<!ELEMENT Languages (Description, Level)>
    <!ELEMENT DesiredJob (\#PCDATA)>
    <!ELEMENT Description (\#PCDATA)>
    <!ELEMENT Duration (\#PCDATA)>
    <!ELEMENT Profession (\#PCDATA)>
    <!ELEMENT Car (\#PCDATA)>
    <!ELEMENT ZipCode (\#PCDATA)>
    <!ELEMENT Town (\#PCDATA)>
    <!ELEMENT Country (\#PCDATA)>
    <!ELEMENT RegionalPreference (\#PCDATA)>
    <!ELEMENT Level (\#PCDATA)>
    <!ELEMENT WorkStatus (\#PCDATA)>
    <!ELEMENT WorkMode (\#PCDATA)>
    <!ELEMENT WorkType (\#PCDATA)>
    <!ELEMENT Salary (\#PCDATA)>
    <!ELEMENT ExperienceLevel (\#PCDATA)>
    <!ELEMENT CarRequired (\#PCDATA)>
```

XML instance for computer scientist:

```
<RSDS>
<Profession>Computer Scientist</profession>
<Profession>Mathematician</profession>
<Experience>
  <ExperienceLevel>expert</ExperienceLevel>
  <Profession>Computer Scientist</profession>
  <Profession>Mathematician</profession>
</Experience>
<CarRequired>false</CarRequired>
<Location>
  <ZipCode>81541</ZipCode>
  <Town>Munich</Town>
  <Country>Germany</Country>
</Location>
<Requirements>
  <GeneralRequirement>
    <Description>soft skills</Description>
    <Level>very good</level>
  </GeneralRequirement>
  ...
</Requirements>
<Description>We are looking for...
</Description>
<WorkStatus>employed</WorkStatus>
<WorkMode>in office</WorkMode>
<WorkType>fulltime</WorkType>
<Salary>40.000 Euro</Salary>
</RSDS>
```

of unemployed persons (currently, about 3.9 millions) just by one day on average will save the German government more than a hundred million dollars a year.

In the HRNET system architecture each company supplies its open positions to a designated GUI-Agent, which has the role of a requester agent in the system. The GUI-Agent queries the matchmaker by sending to HRNET the description of the open position which should be filled.

The service repository of HRNET consists of a collection of data sources. Most of them are databases wrapped by a database wrapper agent. One of them is the central database of the Office for Labor Exchange of the German government, in which all currently unemployed persons in Germany are stored. Another one we used is the Siemens AG internal employee database. Further databases can be easily integrated.

If the number of applicants exceeds a certain limit in a database, the database wrapper agents supplies only a *preselection* of profiles to the matchmaker. This preselection eliminates all profiles which do not match any of the values in $\texttt{Profession}^{+}$ of a given job offer (RSDS instance). In HRNET, the RSDS and OSDS schemes are, as required by GRAPPA, converted to XML-DTDs which are considered as the document classes of these types. The XML-DTD of RSDS, together with an instance, is shown in Table 1.

Besides the generic basic types and distance functions, HRNET uses customized basic types including RegionalPreference, Level and WorkStatus. The default distance functions for these types are defined using distance matrices and exact matching functions. The distances for complex types are computed by using the Hausdorff distance and weighted averages.

Table 2. HrNet matchmaker results

job offering (request) / quality of k-th best match	$k =$ 1	5	10	20	50	100
r_1 : Computer scientist	61 %	45 %	39 %	33 %	23 %	13 %
r_2 : Bus driver	72 %	68 %	61 %	52 %	43 %	30 %
r_3 : Anesthesia male nurse	40 %	23 %	14 %	12 %	10 %	8 %
r_4 : Clerk	36 %	35 %	31 %	26 %	22 %	12 %
r_5 : Truck driver	79 %	70 %	65 %	60 %	55 %	40 %
r_6 : Haircutter	48 %	46 %	39 %	36 %	25 %	12 %
r_7 : Taxi driver	81 %	79 %	75 %	60 %	43 %	20 %
r_8 : Interpreter	52 %	47 %	38 %	34 %	28 %	12 %
r_9 : Children nurse	68 %	67 %	57 %	51 %	29 %	14 %
r_{10} : Engine fitter	59 %	40 %	36 %	29 %	21 %	15 %

3.2 Experiments

In this section, we give a sample of the set of experiments that we have conducted with the HrNet system. It appeared that in these experiments, the HrNet matchmaker performed quite well and was ranking the job applicants (i.e., `OSDS` instances) realistically.

Precison. In the first experiment, we considered ten different job offerings (requests) $r_1, \dots, r_{10}$, which were supplied to a GUI-Agent for querying the matchmaker. Table 2 shows the results for a k best matches (i.e., nearest neighbors) query, where for each k the quality of the last (worst) among the k matches is reported. The quality is the similarity between the request r_i and the applicant profile (offer) o measured by $1 - d(r_i, o)$, where $d(r_i, o)$ is the (normalized) distance value. Request r_1 is the computer scientist instance of `RSDS` shown in Table 1; the other requests instantiated `RSDS` to jobs in different areas.

It is, of course, difficult to judge the quality of matchings computed by the HrNet matchmaker to the one of a human matchmaker, and in particular whether it computes "human like" rankings. Rankings compiled by a human matchmaker may be subjective, and different human experts may come up with different rankings. However, inspection has shown that among a larger set of profiles, the best candidate singled out by the matchmaker is the same one would manually select.

Recall. In a further experiment, we modified the data repository by adding two further `OSDS` instances that should match the request r_1 intuitively high. One of them was intuitively an exact match (quality 100%), and the other one was a profile which intuitively supported more desired properties than the previous best match, which was 61%. As expected, the exact match was the new best best and had a score of 100%. The second best match was the other addition to the data repository. It obtained a significantly better score (85%) than the previous best match.

4 Related Work

[8] considered matchmaking in the context of emerging information integration technologies, where potential providers and requesters send messages describing their capabilities and needs of information (or goods). They presented two matchmakers: COINS (COmmon INterest Seeker), which is based on free text matchmaking using a distance measure from information retrieval [10], and SHADE (SHared DEpendency Engineering), which uses a subset of KIF [6] and a structured logic text representation called MAX [7]. While COINS aimed at e-commerce, SHADE aimed at the engineering domain.

Complementing the theoretical work in ([2,3]), Sycara and coworkers addressed the matchmaking problem in practice. They developed and implemented the LARKS matchmaker (LAnguage for Advertisement and Request for Knowledge Sharing) described in [13,12]. In LARKS, the matchmaking process runs through three major steps: (1) Context matching, (2) syntactical matching, and (3) semantical matching. Step 2 is divided into a comparison of profiles, a similarity matching, and a signature matching. Compared to previous approaches, LARKS provides higher expressiveness for service descriptions. Like those, however, LARKS has a static scheme for service descriptions, which restricts its application to agents that comply with this fixed description format.

In the context of electronic auctions, [16] introduce a service classification agent which has meta knowledge and access to nested ontologies. This agent dynamically generates unique agent and auction descriptions which classify an agent's services and auction subjects, respectively. A requester obtains from it the name of the best auction to its needs.

In IMPACT [1,11], so called Yellow Pages Servers play the role of matchmaker agents. Offers and requests are described in a simple data structure which represents a service by a verb and one or two nouns (e.g., *sell:car*, *create:plan(flight)*). The matchmaking process computes the similarity of descriptions from shortest paths in directed acyclic graphs that are built over the sets of verbs and nouns, respectively, where edges have weights reflecting their distance.

5 Conclusion

In this paper, we have considered the problem of matchmaking given that service descriptions are complex objects, formulated in a rich language. Furthermore, we have presented various methods for computing distance values between complex objects and the GRAPPA framework, which can be used for building matchmaking facilities. As the HRNET application has shown, GRAPPA is an attractive tool for developing application-specific matchmakers.

Our ongoing and future work comprises several issues. One is to exploit the properties of metric distance functions and to design algorithms which avoid scan the entire service repository. The development of specific distance functions is another issue. Last, but not least, an important issue is to improve the efficiency of the access to the service repository, which currently is a bottleneck of the system. In our future work we also intend to transfer matchmaking algorithms into the multi-attribute auction domain.

An interesting issue is the use of self-trained neural networks in the design of customized distance functions which reflect the judgment of a human expert as close as possible. Hence important future work will be to implement neural networks computing distance functions for information classification via matchmaking. Finally, a full scale implementation of the HrNet prototype and the development of further applications of Grappa complement our research.

References

1. K. Arisha, T. Eiter, S. Kraus, F. Ozcan, R. Ross, and V.S Subrahmanian. IMPACT: A Platform for Collaborating Agents. *IEEE Intelligent Systems*, 14(2):64–72, March/April 1999.
2. K. Decker, K. Sycara, and M. Williamson. Matchmaking and brokering. In *International Conference on Multi-Agent Systems (ICMAS96)*, December 1996.
3. K. Decker, K. Sycara, and M. Williamson. Middle-agents for the internet. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 578–583, August 1997.
4. T. Eiter and H. Mannila. Distance measures for point sets and their computation. *Acta Informatica*, 34:109–133, 1997.
5. T. Eiter, D. Veit, J.P. Müller, and M. Schneider. Matchmaking for Structured Objects. Extended version, manuscript, 2000.
6. M. R. Genesereth and R. E. Fikes. Knowledge Interchange Format, Version 3.0 Reference Manual. Technical Report Logic-92-1, Computer Science Department, Stanford University, June 1992. `http://www-ksl.stanford.edu/knowledge-sharing/papers/kif.ps`.
7. D. Kuokka. *The Deliberative Integration of Planning, Execution, and Learning*. PhD thesis, School of Computer Science, Carnegie Mellon University, 1990.
8. D. Kuokka and L. Harada. Integrating information via matchmaking. *Journal of Intelligent Information Systems*, 6(2/3):261–279, 1996.
9. M. Nodine, W. Bohrer, and A. H. H. Ngu. Semantic brokering over dynamic heterogenous data sources in InfoSleuth. In *Proceedings of the Fifteenth International Conference on Data Engeneering*, pages 358–365, August 1999.
10. G. Salton. *Automatic Text Processing*. Addison-Wesley, 1989.
11. V.S Subrahmanian, P. Bonatti, J. Dix, T. Eiter, S. Kraus, F. Ozcan, and R. Ross. *Heterogenous Agent Systems*. MIT Press, June 2000. (ISBN: 0262194368).
12. K. Sycara, J. Lu, and M. Klusch. Interoperability among heterogenous software agents on the internet. Technical Report CMU-RI-TR-98-22, The Robotics Institute Carnegie Mellon University, Pittsburgh, October 1998.
13. K. Sycara, J. Lu, M. Klusch, and S. Widoff. Dynamic service matchmaking among agents in open information environments. *ACM SIGMOD Record 28 (1), Special Issue on Semantic Interoperability in Global Information Systems*, pages 47–53, 1999.
14. D. Veit. Matchmaking algorithms for autonomous agent systems. Master's thesis, Institute of Computer Science, University of Gießen, Germany, 1999.
15. D. Veit, J.P. Müller, M. Schneider, and B. Fiehn. Spt: Matchmaking for autonomous agents in electronic marketplaces. In *Proceedings of the Fifth International Conference on Autonomous Agents*, 2001.
16. P. Weinstein and W. Birmingham. Service classification in a proto-organic society of agent. In *Proceedings of the IJCAI-97 Workshop on Artificial Intelligence in Digital Libraries*, 1997.
17. G. Wiederhold. Intelligent Integration of Information. In *Proceedings of ACM SIGMOD Conference on Management of Data*, pages 434–437, Washington, DC, 1993.

Prototyping Data Warehouse Systems

Thanh N. Huynh[1] and Josef Schiefer[2]

[1]Institute of Software Technology
Vienna University of Technology
thanh@ifs.tuwien.ac.at
[2]Watson Research Center, IBM
josef.schiefer@us.ibm.com

Abstract. The process of developing and evolving complex data warehouse systems is intrinsically exploratory in nature. Data warehouse projects are notoriously difficult to manage and many of them end in failure. One explanation for the high failure rate is the lack of understanding of the requirements and missing proof-of-concepts for a decision support for the knowledge workers of an organization. In this paper, we present a prototyping approach for data warehouse environments. We show how prototyping can support an incremental and iterative requirement development. We introduce different types of prototypes and illustrate their applicability for data warehouse systems. Finally, we show how to develop data warehouse prototypes with the BEDAWA tool, which supports a rapid development of prototypes by automatically generating the mass sample data for a data warehouse system.

1. Introduction

Prototypes can be used for data warehouse systems to explain and answer open questions of stakeholders. The idea behind prototyping is to cut down on the complexity of implementation by eliminating parts of the full system. It is a technique to reduce the risks of failed data warehouse implementation or stakeholder dissatisfaction. A prototype can be used as early feedback from the users ensuring that the data warehouse team properly understands the requirements and knows how best to implement them.

Even with extensive requirement elicitation, analysis, and documentation practices, some portions of the data warehouse requirements might still be uncertain or unclear to either the stakeholders or the data warehouse team. If these problems are not corrected, an expectation gap between the stakeholders will result. Prototyping allows the envisioned data warehouse system to become tangible, brings use cases to life, and closes the gaps to the different understanding of requirements. Furthermore, users are generally more willing to try out a prototype than to read the complete requirements specification.

Any prototype is designed to focus on only certain aspects of the system, compromising or ignoring other aspects. There are three major dimensions of compromises *system functionality, system attributes, and user interface* [4].

Data warehouse prototypes can focus on one or more of these dimensions, while ignoring the others. For instance, some prototypes focus purely on performance (which represents a system attribute), ignoring user interface and functionality issues. Other prototypes might be developed to discover an optimal user interface for

executives, approximating other aspects only to the degree needed to allow it to serve as driver for sponsorship.
The remainder of this paper is organized as follows. In section 2, we discuss the contribution of this paper and related work. In section 3, we present prototyping activities for data warehouse environments and shows how they are interwoven with the requirement process. In section 4, we discuss different types of prototypes. In section 5, we introduce the BEDAWA prototyping tool and show, how it can be used to generate representative sample data for data warehouse systems. Finally, in section 6 we present our conclusion.

2. Contribution and Related Work

Floyd provides in [3] a high-level characterization of prototyping approaches, tools and techniques, and a discussion of problems and prospects of prototyping. Floyd's high-level characterization includes the distinction between horizontal and vertical prototyping. His characterization of prototyping approaches is based on a distinction between exploratory, experimental and evolutionary prototyping.

In this paper, we discuss different type of prototypes for data warehouse systems by using Floyd's prototyping characterization. Furthermore, we also introduce the idea of using prototyping scenarios, which are a combination of horizontal and vertical prototypes.

[17] argues that using measurable objectives (benchmarks) for user performance and attitudes provide an objective way to assess the quality of the prototype. In [5], Huynh proposes a tool, namely BEDAWA, which is able to generate statistical sound, familiar, complete sample data for data warehouse and OLAP systems. The tool allows building sample data for benchmarking applications or various types of prototypes.

[1] introduces the DBGEN tool, which can be used to generate sample data for AS3AP or TPC-D benchmark in flat files. [16] and [7] present an approach for generating sample data to evaluate database systems. However, the presented approach generates sample data manually, which makes it less useful for prototyping purposes.

[10] introduces the easyREMOTE^{DWH} approach, a comprehensive requirements engineering framework for data warehouse systems. easyREMOTE^{DWH} includes a requirements process, which allows an iterative and incremental requirements development. We use this model as foundation and show, how prototyping activities are interwoven with other requirement activities.

To our knowledge, no approach for prototyping has been published yet, which considers the characteristics of data warehouse systems (large, evolving system, heterogeneous user community, complex data structures, large amount of data etc.). Traditional prototyping tools lack the ability to support the generation of complex data schemas and large amounts of data, which is typical for data warehouse environments.

3. Requirements Process and Prototyping Activities

In this section we discuss, how prototyping activities are interwoven with the requirements process. Fig. 1 shows an extract of the requirement activities in the easyREMOTE^{DWH} process [10], and how they are linked by their deliverables. The deliverables are shown as moving from one activity to the next. In the "Requirements Gathering, Elicitation" step data warehouse requirements are discovered, which are made rigorous by writing them down in a prescribed manner. Afterwards, the requirements are inspected in the "Requirements Refinement" to determine if they are accurate enough to be added to the final requirement specification draft. Any rejects are sent back to the originator, who probably takes the requirements back to the requirements gathering and elicitation activity for clarification and further explanation.

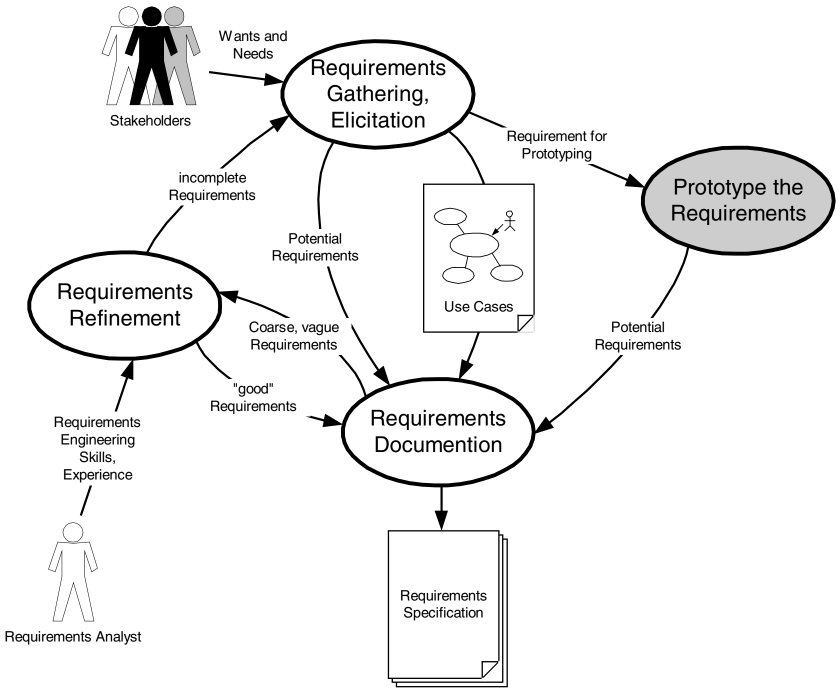


Fig. 1. Requirements Process for Data Warehouse Systems (adapted from [10])

If requirements are still uncertain or unclear to either the stakeholders or the data warehouse implementation team, the development of prototypes can facilitate the management of stakeholder expectations (e.g. by a proof-of-concept) or technical concerns (e.g. by a proof-of-performance). One key point in Fig. 1 is that the requirements process is iterative and incremental rather than linear. The elicitation, prototyping, refinement, and documentation activities are iterated a number of times. During each iteration new information emerges which may necessitate the modification of already acquired information.

Fig. 2 shows the different stages of developing a prototype of a data warehouse system. The input for the prototyping might be a single requirement or a complete use case.

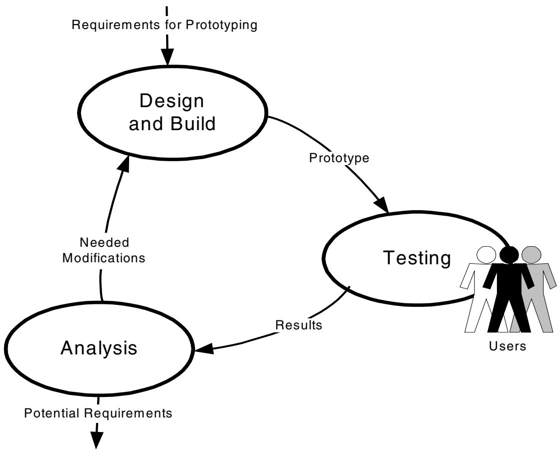


Fig. 2. Prototyping Activities

Design and Build. Designing is mapping the world of the user into the prototype. Design is also the activity of deciding which data warehouse requirements should be modeled and simulated with the prototype, and what should be achieved with the prototype. Furthermore it is important to determine, which type of prototype should be built. After the design stage the data warehouse implementation team builds the modeled prototype.

Testing in the User Environment. Testing involves having the data warehouse users use the prototype as a simulation of their tasks. Users will give feedback of the prototype, where they describe their problems and experience during their work with the prototype. The feedback should also include the results of usability tests.

Analyzing the Results. In this stage the feedback of the users is analyzed. The analysis will uncover new potential requirements for the data warehouse system. If the results are not satisfying, modifications of the prototype and running more tests should be considered.

4. Prototyping Types

4.1 Horizontal vs. Vertical Prototypes

Horizontal prototypes are prototypes where the user-interface is implemented, but the functionality is missing, or simulated. Horizontal prototypes reduce the level of functionality and result in a user interface surface layer, while vertical prototypes reduce the number of features and implement the full functionality of those chosen (see Fig. 3) [3].

Horizontal Prototypes. Horizontal prototypes display the facades of user interface screens from the data warehouse system, possibly allowing some navigation between them, but they do not show real data or contain little or no real functionality. The information that appears in response to a data warehouse query is faked or static, and report contents are hard-coded. A horizontal prototype does not actually perform any useful work, although it looks like it does.

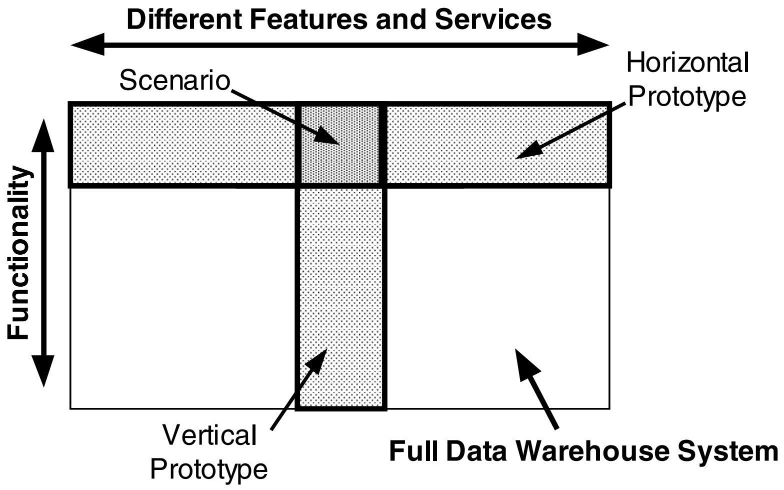


Fig. 3. Horizontal Prototypes, Vertical Prototypes, and Scenarios

The simulation is often enough to give the users a feeling for the data warehouse system and lets them judge whether any functionality is missing, wrong, or unnecessary. The prototype represents the concept to the data warehouse team of how a specific requirement or use case might be implemented. The user's evaluation of the prototype can point out alternative courses for a use case, new missing process steps, previously undetected exception conditions, or new ways to visualize information.

Vertical Prototypes. Vertical prototypes are also known as structural prototypes or proof of concepts. They implement a slice of the data warehouse functionality. Vertical prototypes are developed, when uncertainty exists whether a proposed architectural approach is sound or when certain data warehouse policies should be optimized (i.e. access policies, security policies, data quality policies), or the data warehouse schema should be evaluated (i.e. performance evaluation), or critical timing requirements (i.e. time for data warehouse queries) should be tested. Vertical prototypes are used more to reduce risks during the data warehouse design and implementation than for requirements development. They are generally constructed with the proposed tools of the data warehouse system (database, data staging tools, metadata management tools etc.) in the operating environment to make the results meaningful. Prototype tools, like the BEDAWA tool, can help to construct realistic vertical prototypes.

Scenarios. Scenarios can be seen as a combination of horizontal and vertical prototyping by reducing both the level of functionality and the number of features and services. Since scenarios are small and cheap, they can be applied and changed more frequently. Therefore, scenarios are a way of getting quick and frequent feedback from users. Scenarios can be implemented as paper mock-ups [11], [12] or in simple prototyping environments that may be easier to learn than more advanced programming environments. This is an additional savings compared to more complex prototypes requiring the use of advanced software tools.

4.2 Throwaway vs. Evolutionary Prototypes

Before a prototype is built, the decision has to be made, whether the prototype will be discarded after evaluation or will eventually evolve into a portion of the final data warehouse system. This decision must be explicit and well communicated.

Throwaway Prototypes. The throwaway prototype is most appropriate when you face uncertainty, ambiguity, incompleteness, or vagueness in the data warehouse requirements [2]. These issues have to be resolved to reduce the risk of proceeding with the data warehouse implementation. Because throwaway prototypes are discarded after they have served their purpose, they should be built as quickly and cheaply as possible. When throwaway prototypes are built, solid software construction techniques can be ignored. The emphasis is on a quick implementation and modification over robustness, reliability, performance, and long-term maintainability. For this reason, you must not allow the code from a throwaway prototype to migrate into a final data warehouse system. Development teams often tend to recycle existing throwaway prototypes. A good way to avoid this problem is to use different tools and a different programming language for the prototype construction.

Evolutionary Prototypes. Evolutionary prototypes provide a solid architectural foundation for building the data warehouse system incrementally as the requirements become clearly defined over time. In contrast to the throwaway prototyping, an evolutionary prototype must be built with robust, high-quality code from the beginning. Therefore, an evolutionary prototype takes longer to build than a throwaway prototype that addresses the same functionality. As evolutionary prototype must be designed for easy growth and frequent enhancement, it is important to emphasize the system architecture and solid design principles. For data warehouse systems, evolutionary prototypes are particularly interesting for simulating data models. If a tested data model fulfills the requirements for robustness, extensibility, and performance, it can be used as the foundation for the final data warehouse implementation.

5. BEDAWA – Prototyping Tool

When building a prototype for a data warehouse system, one major problem has to be solved - the generation of data for the data warehouse. It can very difficult to use production data for this purpose, since the data model of the data warehouse system can be very different from the data model of production systems, and consequently, data transformations would be necessary. Once a data basis is established, any prototype can be built using this data.

The BEDAWA tool offers support for this problem. It allows developers of data warehouse prototypes to produce sample data suitable for their prototype implementation. By specifying the characteristics of the sample data and by testing the sample data generation results, they are able to repeatedly develop sample data for their needs. In this section we give an overview of the BEDAWA approach and show how to generate mass sample data for prototyping purposes.

5.1 Introduction of the BEDAWA Approach

The lack of sample data for data warehouse or OLAP systems usually makes it difficult for organizations to evaluate, demonstrate or build prototypes for their systems. However, the generation of representative sample data for data warehouses is a challenging and complex task. Difficulties often arise in producing familiar, complete and consistent sample data on 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 sample data generation process is usually a complex, iterative process that begins with a modeling step, and ends with the generation of the desired sample data. To get representative sample data that is familiar, consistent, scalable, large amount and that reflects various degrees of freedom, the BEDAWA tool is based on a statistical model that will be introduced in the next sections. The statistical correctness of the model provides a framework to define relationships between dimensions and facts in the context of a star schema, the most popular schema for designing and building a data warehouse [8]. The tool provides abilities to define and generate sample data of any size in order to reflect a real world situation. Furthermore, the tool is able to integrate existing external data sources for the sample data generation.

5.2 Statistical Model of BEDAWA

According to the statistical point of view, relationship between dimensions and a fact can be presented as follows:

$$\mathbf{y} = \mathbf{f}(\delta_1, \delta_2, ..., \delta_x) + \varepsilon \qquad \mathbf{(1)}$$

where:

y	: a value of the fact
$\delta_1, \delta_2, ..., \delta_x$	: numbers presenting the effect of dimension members to the fact y
ε	: error value

That means, based on relationship between facts and dimensions of a fact table, a statistical model can be applied to present that relationship stated in formula (1). Besides the calculation of fact values, the statistical model includes a distribution process, which takes care of the task to distribute dimension members in the fact table according a proper statistical distribution.

For example, a fact value can be created by using a linear model as follows [13]:

$$\mathbf{y}_{\mathbf{ijk...}} = \mu + \mathbf{c}_i\alpha_i + \mathbf{c}_j\beta_j + \mathbf{c}_k\gamma_k + ... + \varepsilon_{\mathbf{ijk...}} \qquad \mathbf{(3)}$$

where:

$\alpha, \beta, \gamma, \ldots$	: 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,
$c_i, c_j, c_k, \ldots$	: scalar constants.
i, j, k, ...	: index numbers that are in range [1..n], [1..m], [1..p], ..., respectively,
μ	: average mean of the fact,
$\alpha_i, \beta_j, \gamma_k, \ldots$	: fact effect values of elements ith, jth, kth of dimension fact effect value sets α, β, γ,..., respectively,
$\varepsilon_{ijk\ldots}$	: error value.
$y_{ijk\ldots}$	: a value of the fact.

5.3 Sample Data Generation Process

Fig. 4 shows the sample data generation process of BEDAWA. The first two steps (grayed boxes) are considered as preparation steps. The major purpose of these steps is getting statistical parameters for the sample data. The next steps cover the modeling 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 statistical parameters and data schemas of the data warehouse are used to model the sample data architecture [5]. In the sample data generation step, the modeled sample data are automatically generated.

Users or applications are able to control the sample data generation. That means, depending on the result of a run of a sample data generation, possible iterations of the presented steps can be necessary. If for instance the result of the sample data production does not fulfill the requirements for a prototype, the user will have to go back to the sample data modeling stage to model the sample data correctly. The user iterates between these steps until the produced sample data is satisfying. A further explanation of the sample data generation process is beyond the scope of this paper. A detailed description can be found in [6].

5.4 Generating Sample Data for Prototypes

Builders of horizontal prototypes for data warehouse systems need the complete data structures of the data warehouse with or without data. On the other hand, for vertical prototypes, only a part of data structures is needed, however with a realistic amount and quality of data. By using the BEDAWA tool, both types of prototypes can be built. The tool provides a rich set of functionality for modeling the data structures of a data warehouse system and for describing the quality of the desired sample data.

Following we show how to built a prototype with the star schema shown in Fig. 5. This star schema is the well-known grocery sample introduced in [8].

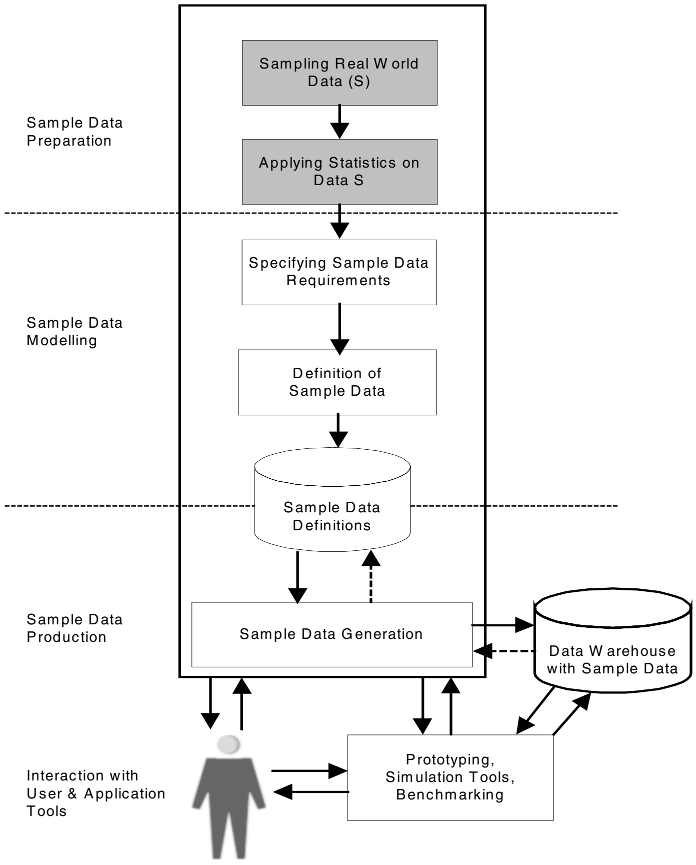

Sampling Real World Data (S)
Applying Statistics on Data S
Sample Data Preparation
Specifying Sample Data Requirements
Sample Data Modelling
Definition of Sample Data
Sample Data Definitions
Sample Data Production
Sample Data Generation
Data Warehouse with Sample Data
Interaction with User & Application Tools
Prototyping, Simulation Tools, Benchmarking

Fig. 4. Process for Generating Sample Data for a Prototype

The generation process of this star schema includes the following steps:

1) Definition of *data sources*. Data sources are used to identify any necessary data for the sample data generation of the prototype.

2) Definition of *dimensions*. Dimension definitions describe the dimension structures of the modelled sample data. For our example, we define the dimensions ProductDimension, StoreDimension and TimeDimension. The dimension data can be either automatically generated or manually copied from available data sources.
Fig. 6 shows the definition of the StoreDimension. A list of attributes is defined, which is described by name, data type, size, and number of data items. Furthermore, each dimension can be constructed based on various hierarchies, e.g., for TimenDimension dimension we can define two hierarchies as follows: day•month•year or dayOfWeek•Week•year.

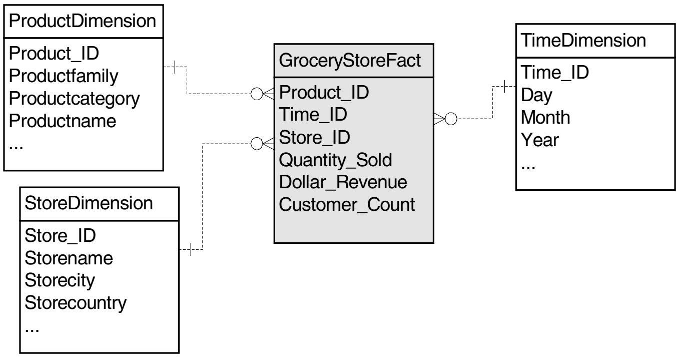
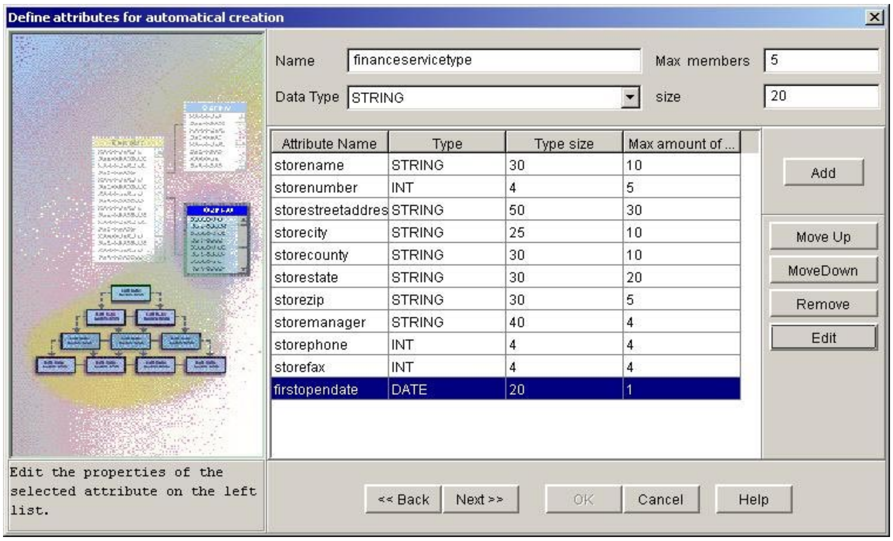


Fig. 5. Star schema

3) Definition of *statistical parameters* for the dimensions. The statistical parameters include distributive information, and fact effect values. Fact effect values quantify each relationship between a dimension and a fact. Distributive information is used to specify the distribution of dimension members in the fact tables for the sample data. The distribution can be statistical distribution or user-defined distribution parameters.


Define attributes for automatical creation

Name: financeservicetype
Max members: 5
Data Type: STRING
size: 20

Attribute Name	Type	Type size	Max amount of ...
storename	STRING	30	10
storenumber	INT	4	5
storestreetaddres	STRING	50	30
storecity	STRING	25	10
storecounty	STRING	30	10
storestate	STRING	30	20
storezip	STRING	30	5
storemanager	STRING	40	4
storephone	INT	4	4
storefax	INT	4	4
firstopendate	DATE	20	1

Edit the properties of the selected attribute on the left list.

Fig. 6. Defining store dimension

4) Definition of the *facts* for the fact table. Each fact is presented by a statistical model, which represents the relationship between fact values and members of

dimension components effecting to the fact. For our example, we defined following three facts illustrated in Table 1.

Table 1. Fact Definitions

Fact Name	Fact Definition	Error value
F(quatity_sold)	253 + FactEffect(time2_fe1) + FactEffect(product_fe1) + FactEffect(promotiondimension_fe1)	Random in [-10,10]
F(dollar_revenue)	1453 + FactEffect(time2_fe2) + FactEffect(product_fe2) + FactEffect(promotiondimension_fe2) + FactEffect(storedimension_fe1)	Normal distribution $\mu = 10$ $\sigma = 2$
F(customer_count)	42 + FactEffect(time2_fe3) + FactEffect(promotiondimension_fe3)	Random in [-5, 7]

5) Defining the *fact table schema*. A fact table consists of various facts defined in the previously step and dimensions, which should be included in the fact table. E.g., the Grocery_Store fact table of our example includes the 3 facts quantity_sole, dollar_revenue, and customer_count and the 3 dimensions ProductDimension, StoreDimension, and TimeDimension.

6) *Generation of sample data*. In this step the user specifies the amount of sample data to be created and starts the generation process. Depending on the type of prototype (horizontal vs. vertical), the users can decide how large the fact table should get.

As in the previous steps shown the BEDAWA sample data generating process supports a flexible specification of the data warehouse metadata (dimension attributes, dimension hierarchies, fact, fact schema etc.) and business data. As result, such sample data is a sound basis for building prototypes for a proof-of-concept or proof-of-performance.

6. Conclusion

The expectation of organizations that more or better requirement and development tools would be a remedy to all problems of building data warehouse systems cannot be fulfilled. Case studies about prototyping in large industry projects show that the central problem to be solved is the difference between application knowledge and information processing knowledge [9]. This difference is clearly visible at the gap between the expectation of stakeholders towards the envisioned system and the resulting data warehouse system.

In this paper, we stressed the importance of building effective prototypes for closing this gap. We have shown how prototyping activities are part of the requirement process and how they facilitate iterative and incremental requirements development. We discussed different types of prototypes, by using the high-level characterization of Floyd. We introduced the BEDAWA tool as prototyping tool for generating sample data for data warehouse systems. The BEDAWA provides a statistical model for the specification of sample data and allows the generation of sample data of various amount and type for prototyping and benchmarking purposes.

References

[1] Bitton, D.; Orji, C., Program Documentation for DBGEN, a test database generator, University of Illinois at Chicago.

[2] Davis, Alan M., Software Requirements: Objects, Functions, and States, PTR Prentice Hall

[3] Floyd, C. A, A Systematic Look at Prototyping.
In: Bude et al. (eds) approaches to Prototyping Springer Verlag, 1984

[4] Goldman, N.M.; Narayanaswamy, K., Software Evolution through Iterative Prototyping 14th Int. Conf. on SW Eng., IEEE, 1992

[5] Huynh, T.; Nguyen, B.; Schiefer, J.; Tjoa, A M., BEDAWA - A Tool for Generating Sample Data for Data Warehouses. Data Warehousing and Knowledge Discovery, 2nd Int'l Conf., Dawak 2000, Greenwich, UK., Springer-Verlag.

[6] Huynh, T.; Nguyen, B.; Schiefer, J.; Tjoa, A M., Representative Sample Data for Data Warehouse Environments ADVIS2000, Turkey

[7] Informix Redbrick Warehouse, TPC-D and its relevance, Informix Corporation

[8] Kimball, R. (1996). The Data Warehouse Toolkit, John Wiley & Sons Inc.

[9] Lichter, H.; Schneider-Hufschmidt, M.; Zuellighoven, H. Prototyping in Industrial software Projects - Bridging the Gap Between Theory and Practice, IEEE Trans. on SE, Vol 20, No 11, pp 825-832, 1993

[10] List, B.; Schiefer, J.; Tjoa, A M., Use Case Driven Requirements Analysis for Data Warehouse Systems, Data Warehousing 2000, Friedrichshafen

[11] Nielsen, J., Prototyping user interfaces using an object-oriented hypertext programming system, Proc. NordDATA'89 Joint Scandinavian Computer Conference Copenhagen, Denmark

[12] Nielsen, J., Paper versus computer implementations as mockup scenarios for heuristic evaluation, Proc. INTERACT'90 3rd IFIP Conf. Human-Computer Interaction Cambridge, UK

[13] Rao, C. R.; Toutenburg, H., Linear Models least Squares and Alternatives, Springer-Verlag New York, Inc. 1995.

[14] Schach, S. R., Classical and Object-Oriented Software Engineering, 3rd, IRWIN.

[15] Schiefer, J.; Tjoa, A M., Generating Sample Data for Mining and Warehousing Proc. Int. Conference on Business Information System Springer-Verlag, BIS 99, Poznan

[16] Turbyfill, C.; Bitton, D., AS3AP-An ANSI SQL Standard Scalable and Portable Benchmark for Relational Database Systems. The Benchmark Handbook. J. Gray, Morgan Kaufmann Publishers.

[17] Whiteside, J.; Bennett, J.; Holtzblatt, K., Usability Engineering: Our experience and Evolution. In: Handbook of Human-Computer Interaction Amsterdam, Elsevier, 1988

Information Warehouse for Medical Research

Anne Tchounikine, Maryvonne Miquel, and André Flory

LISI / INSA de Lyon, 20 avenue A. Einstein, 69621 Villeurbanne, France

{atchouni, miquel, flory}@if.insa-lyon.fr

Abstract. Data warehousing imposes itself as an attractive solution for centralizing and analyzing high quality data. In the medical research field, this technology can be used to validate assumptions and to discover trends on large amount of patient data. However, like other scientific complex data, medical data and especially raw sensor data need to be processed before becoming interpretable. The selection of the process mode is a key issue in the physician's reasoning. In our study, we propose a solution to gather data and processes into a single information warehouse. Our solution provides features for loading, modeling and querying the information warehouse. Stored data are multidimensional data (patient identity, therapeutic data…), raw sensor data (electrocardiogram, X-ray…) and processes. A prototype has been implemented and is illustrated in the cardiology domain to finalise processes in order to detect heart arrhythmia and acute myocardial ischemia that may lead to sudden cardiac death.

1 Introduction

Medical area, like other modern areas, produces increasingly voluminous amounts of electronic data. Conventional administrative data, therapeutic and diagnostic data, now are completed with complex data devices such as X-ray pictures, echography, electrocardiogram, etc… captured by electronic medical This statement of fact, like in business or retail areas, raised the idea to extract and gather this data lying in heterogeneous and distributed system (HIS, PACS,…) in order to discover useful information. Thus clinical data warehousing becomes a solution to collect, store and manage high quality data for medical studies and researches. So far, many of the settled clinical data warehouses have been designed for socio-economical purposes, i.e. to evaluate and improve healthcare costs to follow hospital managers or administrative authorities demands [3,8,10,16].

An important and rising application field for medical warehousing is medical research. Indeed, one form of the medical research consists in discovering trends or confirming hypothesis over large amount of data captured on selected population of patients. Data warehousing technology then stands as an obviously valuable support. However, clinical warehousing for research or epidemiological purposes requires special constraints. [9] draws a description of research issues posed by the implementation of clinical data warehouses and focuses on the need in advanced modeling features in order to represent the high complexity and specificity of medical data.

Another problem which is barely addressed is due to the semantic of medical data, and to the way physicians deal with it. In cardiology area, for instance, the main goal of many researches is to find indicators and descriptors to understand and characterize heart pathologies [1,12]. This goal induces the scientists to develop efficient algorithms (based on signal processing, pattern recognition, statistical methodologies…) to transform the initial data (ECG, representing the twelve leads of a standard electrocardiogram) or a set of data (current ECG, past ECG, biological patients' data,..) into relevant information (data descriptors, risk factors, diagnosis class…) and to validate them on a large scale database. Beyond the difficulties encountered in extracting and modeling medical data, this example emphasizes the difficulty in manipulating, interpreting the data and the need in promoting raw data into useful information. Still, it relies essential for the physicians not to loose the relationship with the original raw data and to be always able to go back to it. Therefore, for a medical warehouse being well-suited for medical research, it should support numeric and/or textual conventional data (administrative, pharmaceutical,… data), as well as complex raw data with the tools or features required for their understanding. These processes are often very hard to elaborate, and one should be able to share this knowledge, and then to allow the user to modify, adapt, and chain them differently according to his purposes or specialty. Thus, these processes should be fully, tightly integrated in the warehouse and be part of it.

In this paper we present a solution, architecture and tools that gather into a single warehouse, conventional multidimensional data, raw data, and software components needed for their manipulation, transformation and/or interpretation. The user manages and displays indistinctly raw or processed data, i.e. relevant information.

Related domains are discussed in section 2. In section 3, we present our proposal and a case study in cardiology field. Section 4 describes the architecture for an information warehouse. Section 5 presents our prototype and illustration using the case study.

2 Related Domains

Data warehouse [7] has became a leading topic in the commercial world as well as in the research community [2, 5, 15]. Until now, data warehouse technology has been mainly used in business world, in retail or finance areas for examples. The leading motivation is to take benefits from the enormous amount of data that relies in operational databases. Therefore, the storing of arranged data extracted from distributed systems in subject oriented warehouses, improves and helps management's decisions. Warehouse data models [14], known as multidimensional models, are designed to represent measurable facts or indicators and the various dimensions which characterize the facts . As an example, in retail area, typical facts are price and amount of a purchase, dimensions being product, store, time and customer. Business applications deal with simple types of data, mainly numbers and text. Applied operations and queries are selection operations, aggregation operations, rolling-over the dimensions levels and are implemented using extended query-languages. When more enhanced features are required, as correlation discovering, or data mining, then they are provided by add-on tools. Then, data warehouses are intended to emphasis relevant indicators for decision making, providing organized, voluminous but simple data, and optimized elementary operations.

In another area, database community has worked for many years on scientific data management [4,13,17]. [4] is a report on an invitational NSF workshop in the 90's that brought together computer scientists and researchers from various fields of the space, earth and life science. This report highlights special recommendations for the design and management of scientific databases. It focuses on the need in managing different types of data, from raw or sensor data to calibrated data (physical corrected data), validated data (filtered data), derived data (aggregated data) up to interpreted data. Focus is also on the need in proposing appropriate analysis operators and user interface suited to the increasing complexity of data and performed algorithms.

Data warehouse experiments have focused on how gathering and sharing data ; scientific DBMS researches have principally focused on optimization of computations over large amount of data. Now, an important addressed issue in scientific DBMS is due to the distributed information sources and users, and platforms heterogeneity and a significant objective is to be able to share pre-existing knowledge on raw data and on derivation semantics [6,17]. Thus we believe that building a framework for a warehouse of scientific data and knowledge could be a solution.

3 Proposal

As argued earlier, processes from scientific domains like signal processing and analysis, pattern recognition, etc…, are to be applied on complex raw data to compute relevant descriptors for the physician. In order to take into account the specific requirement of the medical research, we propose a solution that gathers data and software components into a same warehouse of *information*.

3.1 An Information Warehouse

The information warehouse we propose contains equally data that are modeled in a conventional multi-dimensional manner (for instance data that are necessary to compute homogeneous groups of patients), raw data (X-ray, ECG….), and software components which are needed to manipulate and interpret the data. Raw data on which software components are applied turns to relevant and valuable information. The physician who explores the warehouse can select in a same way initial raw data, and/or processed data for which he can choose and adapt the calculus mode.

Special requirements for a medical information warehouse are :

- *Advanced type support:* as pointed out by [11] medical data management demands to pay a special attention in the design of a model that suits to medical data specificity: advanced temporal support, advanced classification structure, support for continuously valued data… These very sensible questions constitute a substantial problem and are beyond the scope of this article.

- *Complex raw data management*: conventional data warehouses are designed to powerfully handle simple data such as text or numbers. In the medical domain, many useful data are multimedia, e.g. X-rays, pictures, signals… An information warehouse should provide storage for this type of data, and enhanced features to acquire and visualize raw data.
- *Integration* of *software components*: to become usable, raw data has to be processed by increasingly sophisticated programs from several domains (signal and image processing, statistic methods, pattern recognition, neural network approach …). Data processing facilities must be included in the medical information warehouse in order to specify, select and parameterize processes that transform data into relevant information.
- *Share and re-use of validated processes*: design and validation of a process is a difficult and co-operative task that involves several experts e.g. data acquisition experts (radiologist, nuclear physicians…), software engineers, and information interpretation experts (researchers, clinicians, …). A information warehouse can be an opportunity to share a valuable knowledge by storing validated software components. New processes can be created incrementally chaining validated existing processes. This means that the design of the software components must be modular and generic to be reusable.
- *Processed information management*: in order to easily build new data transformation processes, from raw data, and/or processed data, the processed information must be managed in the same way than the initial stored data. For example, a pre-processed signal set can become the material for statistical studies and its access must be complete openness to the researcher when he uses the information warehouse.

3.2 A Case Study

We now describe briefly a case study from the cardiology domain, which we will use later to illustrate our proposal. New methods of signal processing and quantitative electro-cardiology make the electro-cardiogram (ECG) a privileged material to detect heart arrhythmias and acute myocardial ischemia that might lead to sudden cardiac death. Among the different measurements that are performed on ECG, one of the most studied one is called the QT interval. The QT interval measures the time after which the ventricles are again repolarized (see figure 1).

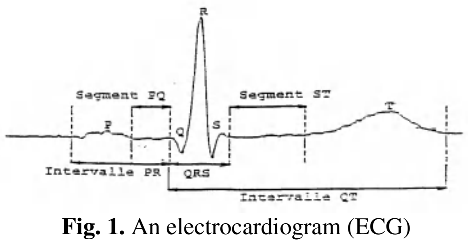


Fig. 1. An electrocardiogram (ECG)

The duration of a QT is mainly influenced by the inverse of the heart rate (RR). The method proposes to model the QT dynamic behavior in function of the history of

RR intervals by means of artificial neural networks. The networks will learn the following non-linear relationship:

$$QT_i = f(RR_i, RR_{i\ 1}, ..., RR_{i\ M+1}, RR_{i\ M}) \text{ where M is a time delay.}$$

For the modeling of QT dynamics, the neural network model is based on a multi-layer perceptron (MLP) architecture. The MLP parameters are the number of hidden neurons and the type of transfer function associated to each neuron. Its learning capabilities are in the weights of connections between neurons and the bias of each neuron.

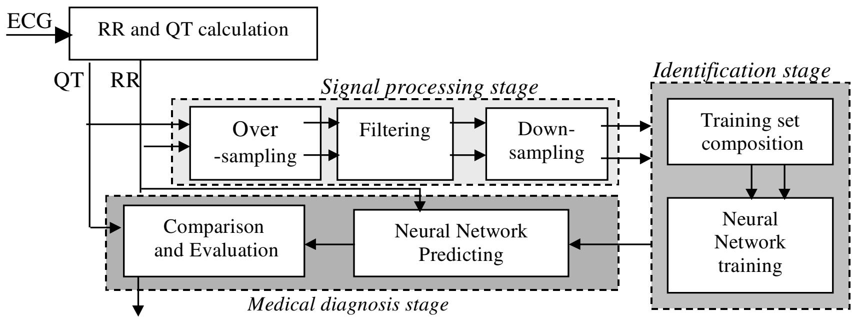


Fig. 2. Simplified view of successive stages to transform ECG data for medical diagnosis

Figure 2 shows the software components used to implement this application and the main stages : signal processing, identification of the Neural Network model and diagnosis elaboration. Before becoming operational for a diagnosis use, the development of neural network models needs several incremental steps to increase the knowledge of the physiological phenomena that operates, to choose the appropriated parameters and to validate the global process. For instance, questions that have to be answered are the validity domain of these models (intra or inter patients, night or diurnal models…), the pertinence of the data preprocessing, the data set for learning and the neural network architecture (number of input and hidden neurons for the model…), or the validity of the diagnosis elaboration. To answer these fundamental research questions, the study must be interactive and iterative. This means that the researcher needs not only to access data but also that he has to select, parameterize the processes to apply. This study highlights the importance of raw data because the preprocessing is not definitively determined and the original signal is indispensable for the result interpretation.

4 Architecture and Tools for a Medical Information Warehouse

Our proposal is an integrated system that implements loading, mapping and querying an information warehouse. Storing is provided by a commercial relational DBMS with multimedia features. Objects that are integrated in the warehouse can be traditional data, provided by operational distributed systems, raw data, acquired with specialised electronic measurement devices, and software components, i.e. processes.

Initial data and processed data are viewed as information. The proposed system (figure 3) is composed of functional modules for data integration, process integration, meta repository management and query interface.

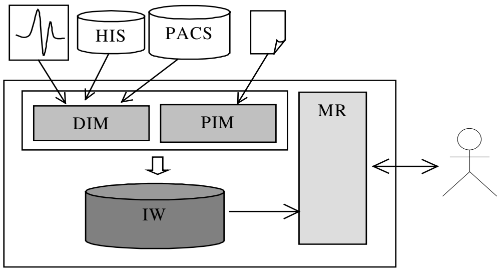


Fig. 3. Architecture

In the following, we describe the different modules. Then, in section 5, we describe our prototype and show, as an example, how the case study described in cardiology area can be integrated and executed using this tool.

4.1 The Data Integration Module

The data integration module (DIM) implements the classical stages of the data warehouse loading: extraction of data from the distributed operational systems, transformation of data and data cleaning. For a Medical Information Warehouse (MIW), the extraction stage consists in extracting data from the HIS including all clinical examinations. Data sources may be distributed and heterogeneous. Therefore, data transformation is the stage that consists in mapping different scheme of data sources into the warehouse schema. Because our choice is to keep as well as possible raw data, the data transformation stage is often reduced to a format change to deal with the possible heterogeneity of the data sources. The cleaning stage is a very important stage because medical researchers need to use data with a very high quality. This stage involves specific procedures for invalid, incompatible or missing data. The data integration module is completely designed with a commercial tool.

Data in the MIW is multidimensional. Data warehouses designers generally use star schema or snowflake schema to represent the multidimensional data model. In the presented case study, a complex snowflake schema is implemented. The central fact entity in the snowflake is the patient, the other entities (heart diseases, pathology family, sex, age, ECG records, RR sequence and QT sequence for each ECG record, signal features for each sequence …). compose the dimensions. Data are stored in a conventional relational database. Then, the researcher can easily extract patients to compose homogeneous population by grouping patients with common features. For example, he can study the population composed of female patients between 25-45 years old with an ischemia pathology and for whom ECG records have particular signal features.

4.2 The Processes Integration Module

The processes integration module (PIM) deals with the design, the loading and the management of software components. A software component is a piece of code, a program or a function that realizes data transformation.

In the MIW system, a data processing software component is either a Tool or a Process. A Tool is an elementary function $Y = f(X, p)$ that describes a calculus. A Tool is defined by:

- a function *f* (data transformation function),
- the set of input variables X of the function *f*,
- the set of output resulting variables Y,
- the set of parameters p whose initial values are known or given by the user at run time of the Tool.

The function *f* is implemented as a piece of code that performs an elementary calculus. From the user point of view, a Tool is a black-box that transforms an input X into an output Y under the control of parameters p. The same yields for the Process who may be viewed as a black-box composed by elementary Tools. A Process is obtained by linking instances of Tools or Processes and as for the Tool, a Process includes input, output and parameters data.

To design a new process, the user first analyses and decomposes his application in elementary functional blocks. Some of these blocks may already pre-exist or need to be created as Tools. Then a Process description consists in describing graphically with an iconic language the links between the Tools or the Processes that compose them.

Let *f* and *g* be 2 elementary tools. A process P can be defined linking *f* and *g* iff Z is part of Y. In this case input of P is X and output is W.

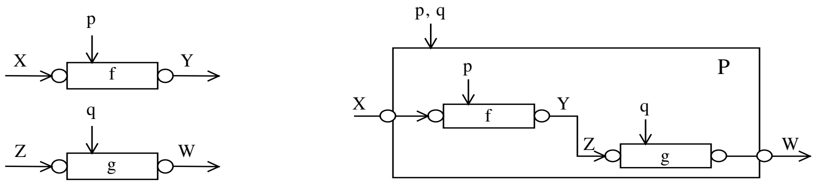


Fig. 4. Tools and processes

4.3 The Meta Repository Module

The meta repository module manages the logical view of initial data and processed information. It is used both for the processes integration and the exploitation of the MIW for medical researches. It gives a common user interface with an unified vocabulary. An Information is defined by:

- a Name (character string) ;
- a Descriptor ;
- and eventually a Value.

The Descriptor structures the data according to the user point of view and introduces semantic knowledge and relations between Information. It is used to describe complex and/or abstract types of the input X and the output Y of the tool or process described above.

A Descriptor is composed of:

- an Identifier (character string);
- a Type (single, list or structure);
- an Interface for the visualisation of the information. If multimedia, the interface can be a complex viewer tool;
- and eventually an Inheritance Link.

For instance, the Descriptor NAME has a character string type, the Descriptor DATE is alphanumeric, HEIGHT and WEIGHT are two Descriptors with numeric type, the Descriptor ECG has a matrix type to store the 12 leads of a standard electrocardiogram. The Descriptor ECG_LIST is defined as a list of ECG Descriptors. All these Descriptors are used in the definition of the Descriptor PATIENT. PATIENT is a Descriptor with a Type structure composed as:

Descriptor PATIENT

Field Name	F*ield Descriptor*
Patient_name	NAME
Creation_date	DATE
Modification_date	DATE
Patient_Date_of_birth	DATE
Patient_height	HEIGHT
Patient_weight	WEIGHT
Patient_sex	CHARACTER
Patient_ECG	ECG_LIST

The Descriptor POPULATION is a list of PATIENT descriptors. With these descriptors, the following information can be defined:

- Mr Smith is an information, with Descriptor PATIENT, and Value (Smith, 01/01/2001, 03/01/2001, 14/12/1954, 1m80, 80kg, M, {ecg1, ecg2}).
- The population IschemicPatients is an information with Descriptor POPULATION, and Value (Smith, John, James).

4.4 Interoperability and Mediation

The inheritance relationships between descriptors are used to enforce consistency rules while linking tools or processes. Let A and B be two elementary Tools (figure 5), Y_a the output of A and X_b the input of B, the connection between A and B is possible if and only if:

- X_b and Y_a have the same Descriptor, or
- Y_a is a list of elements, and each element has the same Descriptor than X_b, or
- The Descriptor of Y_a inherits of the Descriptor of X_b, or
- Y_a is a field of the Descriptor of X_b, or
- X_b is a field of the Descriptor of Y_a.

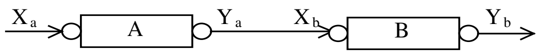


Fig. 5. Connections between pair of Tools A and B

The concept of Descriptors operates at the design step to detect mistakes or anomalies in a connection. For instance, this notion prevents from the interconnection of Processes that use Information in different metric units (miles versus km). In this case, the use of mediators is necessary to adapt the output Information to the input Information (figure 6).

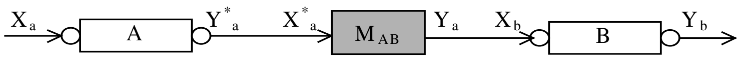
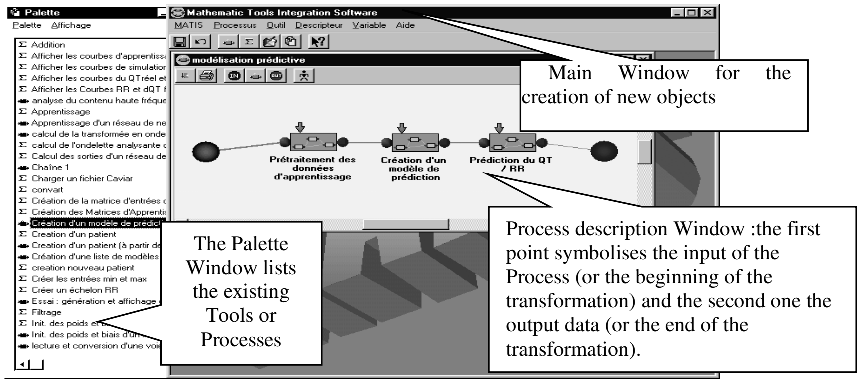


Fig. 6. Use of a mediator Tool

Mediators are used mainly to convert data (basic mathematical operator) or to extract and structure data (adaptation of data views, translators). In the MIW System, mediators are specific Tools but are managed the same way and use the same consistency rules. They provide software and data interoperability.

5 Prototype

The MIW System has been designed following an object-oriented approach and coded in Visual Basic language. The Interfaces associated to the Descriptors are ActiveX controls. For the moment, the functions included in the Tools are coded in Matlab, but extensions to C or C++ programs are allowed by the DLL functionality.


Palette
Palette Affichage
Addition
Afficher les courbes d'apprentiss
Afficher les courbes de simulatior
Afficher les courbes du QTréel et
Afficher les Courbes RR et dQT
analyse du contenu haute fréque
Apprentissage
Apprentissage d'un réseau de ne
calcul de la transformée en onde
calcul de l'ondelette analysante
Calcul des sorties d'un réseau de
Chaîne 1
Charger un fichier Caviar
convart
Création de la matrice d'entrées
Création des Matrices d'Apprenti
Création d'un modèle de prédict
Creation d'un patient
Creation d'un patient (à partir de
Création d'une liste de modèles
creation nouveau patient
Créer les entrées min et max
Créer un échelon RR
Essai : génération et affichage
Filtrage
Init. des poids et
Init. des poids et biais d'un
lecture et conversion d'une voie
Mathematic Tools Integration Software
MATIS Processus Outil Descripteur Variable Aide
modélisation prédictive
IN
OUT
Prétraitement des données d'apprentissage
Création d'un modèle de prédiction
Prédiction du QT / RR

Main Window for the creation of new objects

The Palette Window lists the existing Tools or Processes

Process description Window :the first point symbolises the input of the Process (or the beginning of the transformation) and the second one the output data (or the end of the transformation).

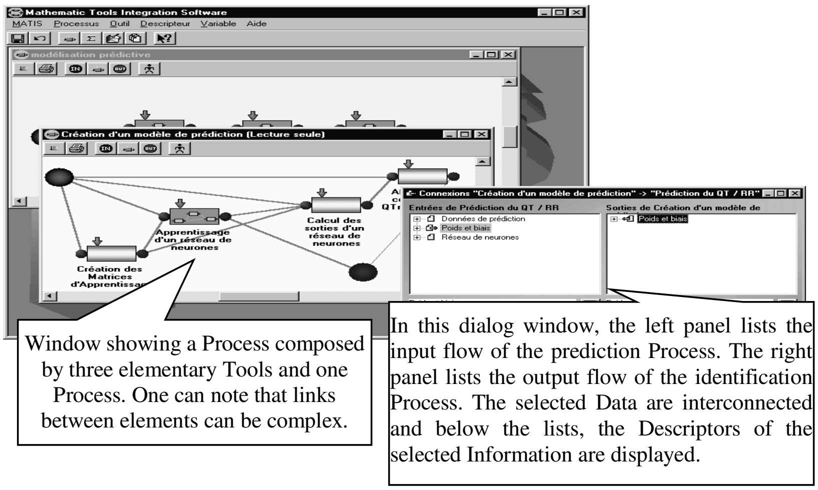
Fig. 7. Main MIW Interface windows

The user interface for the system has been developed to allow the users to have direct access to the MIW objects (Descriptors, Information, Tools and Processes). The interface is based on two main windows (figure 7): the first one lists the existing

Tools and Processes, the second one is the window for the creation of new objects. Inside, a window contains the description of a Process.

Figure 7 displays a typical process for the identification and the use of a predictive model. It is composed of three Processes: the preprocessing of the data, the determination of the parameters under laying the predictive model (model identification) and the process implementing the predictive model. Figure 8 shows the identification Process which is composed itself by three elementary Tools and one Process.

The connections between the Tools or the Processes are graphically represented by links. To simplify the schemes, only one link is drawn between two Processes, even if it represents several logical connections. Figure 8 shows a window for the logical description of a link between the two Processes. Each input or output flow is described by its Descriptors and the user must specify the effective connections. The previous connection rules are then applied for the semantic control of the links.



Mathematic Tools Integration Software
MATIS Processus Outil Descripteur Variable Aide
modélisation prédictive
Création d'un modèle de prédiction (Lecture seule)
Apprentissage d'un réseau de neurones
Calcul des sorties d'un réseau de neurones
Création des Matrices d'Apprentissage
Connexions "Création d'un modèle de prédiction" -> "Prédiction du QT / RR"
Entrées de Prédiction du QT / RR
Données de prédiction
Poids et biais
Réseau de neurones
Sorties de Création d'un modèle de
Poids et biais

Window showing a Process composed by three elementary Tools and one Process. One can note that links between elements can be complex.

In this dialog window, the left panel lists the input flow of the prediction Process. The right panel lists the output flow of the identification Process. The selected Data are interconnected and below the lists, the Descriptors of the selected Information are displayed.

Fig. 8. Logical description of a link

6 Summary

In this paper, we propose a framework for the design, implementation and exploitation of an information warehouse that supports specific medical requirements: handle of complex and multimedia data, storage of raw sensor data, handle of information derived from raw data, storage and election of the processes that implement data derivation. We believe that this framework generalizes well to others scientific fields. The proposed tools supply features for loading software components into the warehouse, for describing metadata and for accessing information through metadata. In the actual version, the description of the universe, i.e. the meta-dictionary, is entirely under the user's responsibility. The use of a domain-specific core ontology could greatly improve this task.

References

[1] H. Behlouli, M. Miquel, J. Fayn, P. Rubel: *Towards Self-Improving NN based ECG classifiers* IEEE Engineering in Medicine an Biology, 18th Annual International Conference, Amsterdam, October 31-November 3, 1996

[2] S. Chaudhuri, U. Dayal: *An overview of Data Warehousing and OLAP technology*, ACM SIGMOD Record 26 (1), 1997.

[3] E.F. Ewen, C.E. Medsker, L.E Duterhoft: *Data Warehousing in an Integrated Health System; Building the Business Case* ACM First International Workshop on Data Warehousing and OLAP (DOLAP), November 7, 1998, Bethesda, Maryland, USA

[4] J.C. French, A K. Jones, J. L. Pfaltz: *Scientific Data Management* SIGMOD Record 19(4): 32-40(1990) special issue on Directions For Future Database Research and Development.

[5] S. Gatziu, M. Jeusfeld, M. Staudt, Y. Vassiliou: *Design and Management of Data Warehouses: Report on the DMDW'99 Workshop*, SIGMOD Record 28(4): 7-10 (1999)

[6] N.I Hachem, K. Qiu, M. Gennert, M. Ward*: Managing Derived Data in the Gaea Scientific DBMS*, Proceedings of the 19th International Conference on Very Large Data Bases, (VLDB'93), Dublin, Ireland.

[7] W.H. Inmon: *Building the Data Warehouse*, Published by: Wiley Computer Publishing, 1996

[8] J. Niinimäki, G. Selén, M. Kailajärvi, P. Grönroos, K. Irjala, J. Forström *Medical Data Warehouse, an investment for better medical care*, Medical Informatics in Europe (MIE'96), Copenhagen, Denmark, August 1996

[9] T.B. Pedersen, C. S. Jensen: *Research Issues in Clinical Data Warehousing* Proceedings 10th Int. Conf. on Scientific and Statistical Database Management (SSDBM), Capri, (Italy,) 1998

[10] T.B. Pedersen, C.S. Jensen: *Clinical Data Warehousing: A Survey*, Proceedings of 7th Mediterranean Conference on Medical and Biological Engineering and Computing, 1998

[11] T.B. Pedersen, C.S. Jensen: *Multidimensional Data Modeling for Complex Data*, Proceedings of the 15th International Conference on Data Engineering (ICDE), 23-26 March 1999, Sydney, Australia, IEEE Computer Society Press

[12] P. Rubel, S. Hamidi, H. Behlouli, JP Couderc, J Fayn, M.C. Forlini, P. Maison-Blanche, M. Miquel, P. Coumel, P. Touboul. *Are serial Holter, Late potential and wavelet measurement clinically useful ?.* Journal of Electrocardiology, n°29, p. 52-61, 1996.

[13] Shoshani, H. K. T. Wong: *Statistical and Scientific Database Issues*. IEEE Transactions on Software Engineering (TSE), Volume 11(10): 1040-1047 (1985)

[14] P. Vassiliadis, T. Sellis: *A Survey of Logical Models for OLAP Databases* SIGMOD Record 28(4): 64-69 (1999)

[15] J. Widom: *Research Problems in Data Warehousing*. Proceedings of the 1995 International Conference on Information and Knowledge Management (CIKM), November 28 - December 2, 1995, Baltimore, Maryland, USA

[16] K. Yetongnon, C. Binquet, C. Quantin, A. Tchounikine, A. Flory, E. Kerkri: *Organisational and Semantic Aspects for Data Warehousing: Application to* Cancerology Proc. Medical Informatics Europe (MIE), Hannover, Germany, august-sept 2000

[17] M. Zemankova, Y.E. Ioannidis: *Scientific Databases - State of the Art and Future Directions.* Proceedings of the 20th International Conference on Very Large Data Bases, (VLDB'94), September 12-15, 1994, Santiago de Chile, Chile.

Risk-Management for Data Warehouse Systems

Robert M. Bruckner[1], Beate List[1], and Josef Schiefer[2]

[1]Institute of Software Technology, Favoritenstr. 9-11 / 188
Vienna University of Technology
A-1040 Vienna, Austria
{bruckner, list}@ifs.tuwien.ac.at

[2]IBM Watson Research Center, 30 Saw Mill River Rd.
Hawthorne, NY 10532
josef.schiefer@us.ibm.com

Abstract. Data warehouse projects are notoriously difficult to manage and many of them end in failure. One explanation for the high failure rate is that managers are not taking prudent measures to assess and manage the risks involved in the data warehouse project. In this paper we introduce the goal-driven risk management approach of the easyREMOTE^{DWH} framework. We classify the risks of a data warehouse system into categories of risk sources and discuss their consequences to the data warehouse project. Further, we show the relevance of risks to the go/no-go decision of the data warehouse project. Finally, we present an approach for documenting data warehouse risks.

1. Introduction

Data warehouse projects are characterized by high decision stakes and high levels of system uncertainty. As such data warehouse projects are difficult to plan and develop with traditional project management methods. Because of the peculiarities and characteristics of data warehouse systems, the project management for such systems is a challenging task. Data warehouse systems are complex, large-scale and constantly evolving systems, which integrate the data of many software systems of an organization. They are used by a wide-range of users with very heterogeneous requirements.

Project managers of data warehouse projects have to cope with new major challenges, which also have a strong impact on the risk management activities. Following characteristics of data warehouse systems make a risk management different to traditional software projects:

User Heterogeneity. Because data warehouse systems are designed and constructed to address concrete business problems or opportunities, the users drive the data warehouse requirements. Data warehouse systems have to cope with a wide heterogonous range of users. The data warehouse requirements have to express all the needs of these users, but also arising conflicts between the user requirements have to be resolved.

Growth and Scalability. A user-driven data warehouse system typically grows fast - in numbers of users and volume of data. For instance ERP sources fuel this rapid expansion because they contain huge amounts of detailed data, which users may want to analyze inside a data warehouse. The constant growth of historical business data can cause bottleneck problems for the data warehouse system.

System Evolution. Data warehouses are high-maintenance systems. Reorganizations, product introductions, new pricing schemes, new customers, changes in production systems, and so on are going to affect the data warehouse. If the data warehouse is going to stay current (and being current is absolutely indispensable for user acceptance), changes to the data warehouse have to be made without delay. Therefore, the data warehouse system has to evolve with the business trends. Risk management has to consider the evolutionary development.

Open System Architecture. Ideally, all data warehouse applications should be supported by a common enterprise architecture that standardizes processes, components, and tools, eliminates redundant activities, and rationalizes business rules and data definitions. Proprietary data warehouse solutions and the lack of supporting existing standards for data warehouse systems can be a major risk factor for a data warehouse project.

System Integration. Data warehouse systems are fed from a wide range of software systems of the organization. Therefore, most of the effort for building a data warehouse system is integrating the data of those software systems.

Many data warehouse risks are reflected by one or more of these characteristics of a data warehouse. This paper presents the easyREMOTE^{DWH} approach, which addresses these risk sources and provides a solid framework for managing these risks including tool support.

The remainder of this paper is organized as follows. In section 2, we discuss the contribution of this paper and related work. In section 3, we classify data warehouse risks and discuss risk consequences to the data warehouse project. In section 4, we discuss the impact of data warehouse risks to the go/no-go decision of a data warehouse project. In section 5 and 6, we introduce the easyREMOTE^{DWH} risk management process and discuss its activities in detail. Section 7 shows templates of the easyREMOTE^{DWH} prototype, which can be used to document risks. Finally, in section 8 we present our conclusion.

2. Contribution and Related Work

Since the 1970s, both academics and practitioners have written about managing risks of software projects [2, 3, 5]. Unfortunately, much of what has been published on risk management is based on studies limited to a narrow portion of the development process of software projects. With a few exceptions [1, 6], there has been little attempt to understand the relative importance of various risks or to classify them in any meaningful way. The characteristics of data warehouse systems are different to normal software projects [8] and therefore, there is a need for a more systematic investigation to identify the major risks that can impact a data warehouse project to classify these risks, and to develop appropriate risk management strategies.

Most risk management approaches do not explicitly support different stakeholder perspectives [3, 4], and those that do, often limit the number of stakeholders and assume that consensus can be reached [9]. Kontio presents in [7] the risk management method Riskit that is based on sound theoretical principles and thus avoids many of the limitations and problems that are common to many other risk management approaches in software engineering. Reskit also includes goals for representing different perspectives of stakeholders. While the Riskit method can be applied in many other domains, such as business planning, technology selection etc., it has been originally developed for software development projects. Therefore, its main features correspond to the risk management concepts and practices required in traditional software projects.

Boehm introduces in [2] a risk management approach for software projects. The projects and environments that were the basis of Boehm's work might not, however, be representative for data warehouse projects. Furthermore, both the organizational and technological landscape has changed considerable since Boehm's work appeared: new organizational forms and systems development environments have emerged, new mechanisms have evolved for acquiring systems (e.g. outsourcing and strategic alliances), and centralized, web-focused systems architectures open new business opportunities for organizations. For all of these reasons, we propose a reexamination of the risk management issue for data warehouse systems.

3. Categories and Consequences of Data Warehouse Risks

Risk management sets forth a discipline and environment of proactive decisions and actions to continuously assess what can go wrong in the data warehouse project, what risks are important to consider, and what strategies to implement to deal with those risks. Data warehouse projects face many kinds of risks besides those related to the project scope and the requirements (see Figure 1).

Dependency on external entities, such as subcontractors, external systems or other related projects, is also a common source of risks for data warehouse systems. The project management is fraught with risks caused by poor estimation, rejection of accurate estimates by managers, insufficient visibility into project status, and staff turnover. A disconnection between the data warehouse and business objectives can be the reason for unfulfilled business returns. The absence of an executive commitment can endanger the sponsorship for the data warehouse system. The failure of constant selling and promoting the data warehouse system internally to the users can result in a rejection of the new system. Escalated project scope by additional data subject areas or including other user groups makes an accurate identification and measurement of deliverables difficult. The bundling of a data warehouse project with other projects can be easier to justify or fund. Because data warehouse systems are complex and involve various kinds of technologies, technology risks can arise.

Lack of knowledge is another source of risk, as with practitioners who have insufficient experience with technologies being used or with the application domain. Successful data warehouse project must respond to changing user community and their information needs. Lack of an ongoing post-implementation support can cause under-utilization of the data warehouse system.

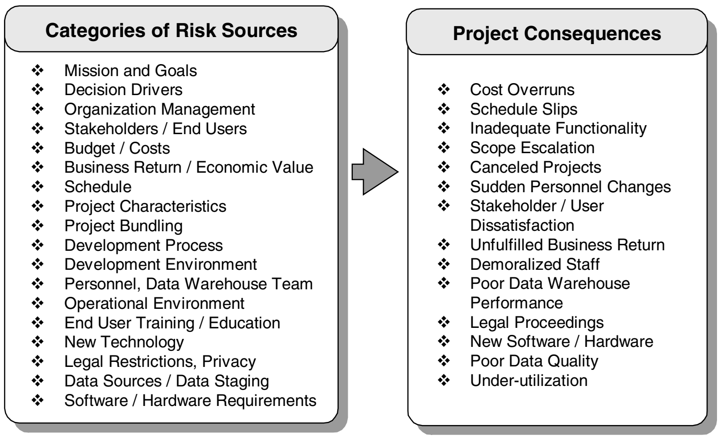


Fig. 1. Risk Sources and Consequences for Data Warehouse Projects

4. Impact of DWH Risks to Go/No-Go Decision

The go/no-go decision of a data warehouse project is strongly influenced by the project risks. Besides the analysis of the risks, also measuring the required effort and determining the values for the stakeholders, makes a composite assessment of the overall worth of the data warehouse system possible.

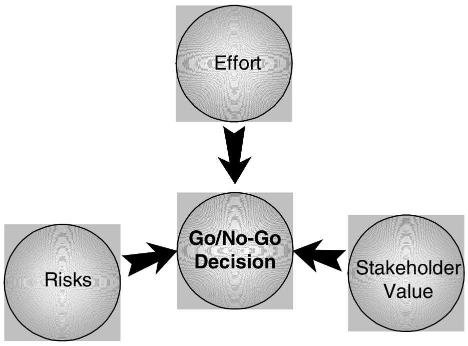


Fig. 2. Go/No-Go Assessment for a Data Warehouse Systems

Figure 2 shows the three relevant dimensions for this assessment. Each of the surrounding bubbles in Figure 2 represents one of the factors that determine if the data warehouse system is worthwhile. The effort bubble can be assessed using function points, use case points or some other size measurement, and represents the cost of construction. The stakeholder value bubble is assessed using metrics such as satisfaction and dissatisfaction ratings. The risks bubble is a measure of the severity of risks determined by the risk analysis activity. Based on the composite assessment of all factors, the decision for continuing or stopping the data warehouse project must be made.

5. Risk Management Activities

Risk management is the application of tools and procedures to contain project risks within acceptable limits [10]. The management of risks for data warehouse systems should provide a standard approach to identify and document risk factors, evaluate their potential severity, and propose strategies for mitigating those risks [11]. Risk management includes the following activities (see Figure 3):

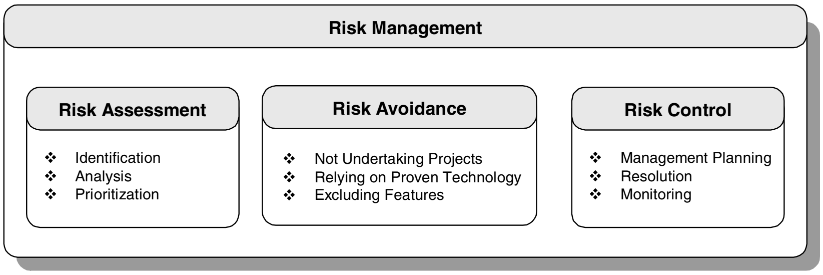


Fig. 3. Risk Management Activities

Risk assessment is the process of examining the data warehouse project to identify areas of potential risks. The risk identification can be facilitated with lists of common risk factors of data warehouse projects. During risk analysis, the potential consequences of specific risks to the data warehouse project are examined. Risk prioritization helps to focus on the most severe risks by assessing the potential risk exposure from each.

Risk avoidance is one way to deal with a risk by simple avoiding risky things. Risks can be prevented by not undertaking certain projects, by relying on proven rather than cutting-edge technologies when possible, or by excluding features that will be especially difficult to implement correctly.

Risk control activities are performed to manage the identified top-priority risk factors. Risk management planning produces a plan for dealing with each significant risk, including mitigation approaches, contingency plans, owners, and timelines. Risk resolution involves executing the plans for mitigating each risk. Finally, a tracking process for resolving each risk item through risk monitoring is essential. It is important to revise the contents and priorities of the risk items periodically to monitor how well the risk mitigation actions are working.

6. Risk Management Process

When a data warehouse team applies risk management, it proactively assesses risks of the data warehouse project continuously, and uses them for decision-making in all phases of the project. They carry the risks forward and deal with them until they are resolved or until they turn into problems and are handled as such. Figure 4 shows the steps of the proactive risk management process of the easyREMOTE^{DWH} framework.

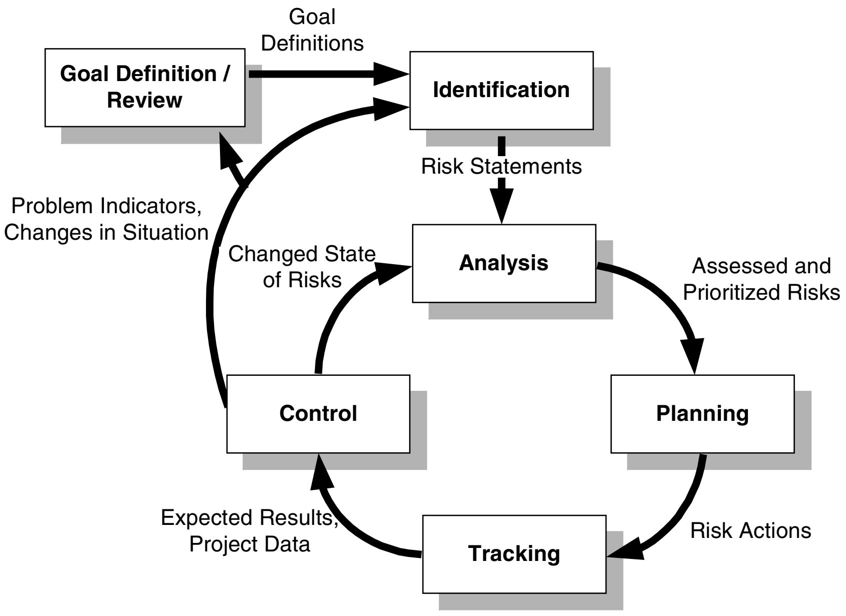


Fig. 4. easyREMOTE^{DWH} Risk Management Process

Goal Definition / Review

This step identifies and defines the project goals and stakeholders, including means to keep goal definitions up-to-date. Goals represent the perspectives of the envisioned data warehouse system. When risk scenarios are defined, their impact to the data warehouse project is specified by the stated project goals. This allows full traceability between risks, goals and stakeholders: each risk can be described by its potential impact on the agreed project goals, and each stakeholder can use this information to rank risks from their perspective.

Risk Identification. Risk identification offers the data warehouse team the opportunities, cues, and information that allow them to reveal major risks before they adversely affect the data warehouse project. Depending on experience, different stakeholders and members of the data warehouse team will see the data warehouse project and the related risks differently. Before the risks of a data warehouse system can be managed, they must be expressed clearly. When stating a risk, the data warehouse team must consider not only a symptom, but also a result. Hence, the statement of risk should include what is causing the situation to arise (risk condition) and the expected result (risk consequence). Therefore, risk statements are made up of a condition and at least one consequence of the condition. The condition is something perceived to be true today (it could be a problem, something neutral, or even a good thing) from which undesirable outcomes for the data warehouse system are expected.

Section 7 shows how risk statements are implemented in the easyREMOTE^{DWH} prototype.

Risk Analysis

Risk analysis ensures that the data warehouse team is working on the right risks. Risk analysis is the conversion of risk data into risk decision-making information. The risk is composed of two factors: (1) risk probability and (2) risk impact. Both factors are used for determining the exposure of a risk.

Risk Probability. Risk probability is the likelihood that an event will actually occur. Risk probability must be greater than zero, or the risk does not pose a threat to the data warehouse project. Likewise, the probability must be less than 100 percent or the risk is a certainty.

Risk Impact. Risk impact measures the severity of adverse affects, or the magnitude of a loss, if the risk comes to pass. If the risk has a financial impact, a value in currency can be the preferred way to quantify the magnitude of loss. The impact to the schedule of the data warehouse project (in man days) can also measure the loss by a risk. Risks can also be quantified by a level of impact where a subjective scale (i.e. from 1 to 5) rates the viability of project success. High values indicate serious loss to the project. Medium values show loss to portions of the data warehouse project or loss of effectiveness.

Risk Exposure. To evaluate a list of risks, the overall threat of each risk needs to be calculated. Sometimes a high-probability risk has low impact and can be safely neglected; sometimes a high-impact risk has low probability and can safely neglected as well. The risks that have high exposure (= high probability and high impact) are the ones worth managing. Therefore, risk exposure is a function of both the probability of incurring a loss due to the risk and the potential magnitude of that loss.

Risk analysis weighs the threat of each risk to help decide which risks merit taking action. Managing risk takes time and effort away from other parts of the data warehouse project, so it is important for the data warehouse team to do only what is absolutely necessary to manage them. The data warehouse team should not spend much time in quantifying risks too precisely. The goal of risk analysis is to differentiate the most threatening risks from those, which do not need to be tackled immediately.

Risk Planning

Risk planning activities turn risk information into decisions and actions. Planning involves developing actions to address individual risks, prioritizing risk actions, and creating an integrated risk management plan. The goals of risk planning include (1) reduction of the probability that a risk will occur, (2) reduction of the magnitude of loss, or (3) change of the consequences of a risk.

In general, groups new to risk management have trouble dealing with more than 10 risks [11]. Therefore, a data warehouse team should consider only a limited number of major risks that must be managed. After the data warehouse team has ranked the risk exposure, it can focus on a risk management strategy and how to incorporate the risk action plans into the overall project plan.

Risk Contingency Strategy. The idea behind a contingency strategy is to have a fallback plan in place that can be activated in case all efforts to manage a risk fail. For example, a new release of a particular ETL (Extraction, Transformation, Loading) tool is needed so that the data of a particular legacy system can be loaded into the data warehouse system, but the arrival of the tool is at risk. The data warehouse team might devise a plan to use an alternate tool or develop an own solution for the loading of the data. Simultaneous development may be the only contingency plan that ensures that the data warehouse system is implemented in time. Deciding when to start the second parallel effort is a matter of watching the trigger value for the contingency plan. Often the data warehouse team can establish trigger values for the contingency plan based on the type of risk or the type of consequence for the data warehouse project that will be encountered.

Risk Tracking

Risk tracking activities include the monitoring of the risk statuses by the data warehouse team, and ensure that actions are taken to mitigate the risks. Risk tracking is essential to an effective action plan implementation. Therefore, risk metrics and triggering events help ensure that the planned risk actions are working. Tracking is the watchdog function of the risk action plan.

An effective way of tracking risks are risk status reports, which show risk exposure vs. time [11]. Risk status reporting can identify four possible risk management situations:

- A risk is resolved, completing the risk action plan.
- Risk actions are tracking the risk management plan, in which case the risk actions continue as planned.
- Some risk actions are not tracking the risk management plan, in which case corrective measures should be determined and implemented.
- The situation has changed significantly with respect to one or more risks and will usually involve reassessing the risks.

Risk Control

Risk control is the last step of the risk management process. After the data warehouse team has chosen risk metrics and triggering events, risks can be similarly controlled as data warehouse requirements. Therefore, risk management melds into the project management processes to control risk action plans, correct for variations from the plans, respond to triggering events, etc. Controlling a risk includes:

- Altering the mitigation strategy when it becomes ineffective.
- Taking action on a risk that becomes important enough to require mitigation.
- Taking a preplanned contingency action.
- Dropping to a watch-only mode at a specific threshold.
- Closing the risk when it no longer exists.

Williams states in [11] that risk information is only useful if it is accessible and easy to understand. Therefore, it is advantageous to hold the risk information in a database. Having the risk data online and accessible to stakeholders and the data warehouse team lets them adjust risk information or enter new risks as soon as they are identified. Furthermore, this information will be a significant benefit to the organization on future data warehouse projects.

7. Documenting Data Warehouse Risks

The risks of a data warehouse system need to be documented and managed in a way that allows the data warehouse team to communicate risk issues to stakeholders throughout the project's duration. In this section we show how to document data warehouse risks by risk statements and risk action statements facilitated by the easyREMOTE^{DWH} prototype, which is implemented in Lotus Notes.

Risk Statements

During the risk identification and risk analysis activities, risk statements gather essential information about data warehouse risks. As mentioned in the previous section, risk statements have a condition-consequence structure. Table 1 shows the sections of a risk statement used by the easyREMOTE^{DWH} prototype.

Table 1. easyREMOTE^{DWH} Prototype: Risk Statement

Section	Description
Risk ID	A number, which uniquely identifies the risk statement.
Risk Category	This section indicates the focus area (i.e. implementation, deployment, enterprise architecture management), or the risk factor category (i.e. decision drivers, schedule, budget/costs) that is used to identify the risk.
Risk Rational	The risk rationale is the reason behind the occurrence of the risk. It contains additional background information that helps to clarify the risk situation.
Risk Condition	A natural language statement describing an existing condition that could possibly cause a loss for the data warehouse project.
Risk Consequence	A natural language statement describing the loss that would occur to the data warehouse project if the risk became certain.
Resolution Needed By	A date, which indicates when the risk has to be at the latest resolved.
Affected Requirement(s) / Use Cases	This section lists requirements and use cases, which are affected by the risk.
Related Risks	This section lists dependent risks.
Risk Status	The current status of the risk. Possible statuses are: Unresolved, Resolved, Being Investigated.
Risk Probability	A percentage number representing the likelihood that the risk condition will actually occur, resulting in a loss.
Risk Loss	The risk loss is the magnitude of impact, if the risk will actually occur. This number could be the loss of money or time, or simply a number between 2 and 10 that indicates relative magnitude (see Table 2).
Risk Exposure	The risk exposure measures the overall threat of the risk to the data warehouse project by balancing the likelihood of the actual loss with the magnitude of the potential loss. The data warehouse team uses the risk exposure to rate and rank the risks.

Risk Action Statement

Risk action statements determine the necessary steps for mitigating a risk or defining a contingency plan for the risk. Table 2 lists the sections of a risk action statement form. The risk action statement form distinguishes between actions for the risk mitigation ("Planned Actions" section) and actions for a contingency strategy ("Risk Contingency Strategy" section).

Table 2. easyREMOTE^{DWH} Prototype: Risk Action Statement

Section	**Description**
Risk Action ID	A number, which uniquely identifies the risk action statement.
Risk ID	The number of the risk statement, which belongs to the risk action statement.
Planned Actions	This section lists planned actions the data warehouse team will take to manage the risk.
Fit Criteria for Risk Action	A fit criterion for risk actions specifies how the data warehouse team will know that the planned actions have been successfully done.
Assigned To	This section lists people assigned to perform the planned action items.
Date To Finish	The date, when each planned action item has to be completed.
Risk Contingency Strategy	This section describes the strategy in the case that the actions planned to manage the risk will not work.
Fit Criteria for Risk Contingency Strategy	The fit criteria for the risk contingency strategy specify how the data warehouse team will know that the risk contingency strategy has been successfully done.
Risk Contingency Strategy Triggers	This section lists the metrics and triggers that the data warehouse system will use to determine when the risk contingency strategy should be put into effect.

8. Conclusion

In this paper we stressed the importance of risk management for data warehouse projects. We have shown that data warehouse projects have characteristics that makes risk management different to traditional software projects. Using a systematic approach for risk management facilitates the control of these risks by the data warehouse team. We have shown different categories of risk sources and risk consequences for a data warehouse project. We introduced the easyREMOTE^{DWH} risk management process and discussed its activities in detail. The risk management approach of easyREMOTE^{DWH} is goal-driven and allows stakeholders to manage the risks from different perspectives. We also introduced templates for capturing risk information, which are implemented by the easyREMOTE^{DWH} prototype.

References

1. Boehm, B. W.: Theory-W software project Management: principles and examples, IEEE Trans. Softw. Eng. 15, 1989
2. Boehm, B.W.: Software Risk Management: Principles and Practices, IEEE, 1991
3. Charette, R.N.: Software Engineering Risk Analysis and Management, Intertext, New York, 1989
4. Garvey, P. R.; Phair, D. J., Wilson, J. A.: An Information Architeture for Risk Assessment and Management. IEEE, 1997
5. Jones, C.: Assessment and Control of Software Risks, Prentice-Hall, 1994
6. Keil, M.; Cule, P. E.; Lyytinen, K.; Schmidt, R. C.: A Framework for Identifying Software Project Risks, ACM, Nov. 1998
7. Kontio, J.: The Riskit Method for Software Risk Management, Computer Science Technical Reports, University of Maryland, 1997

8. List, B.; Schiefer, J.; Tjoa, A M.: Use Case Driven Requirements Analysis for Data Warehouse Systems, Data Warehousing 2000, Friedrichshafen
9. Pandelios, G.: Software Risk Evaluation and Team Risk Management, SEPG Conference, Pittsburgh, PA
10. Wiegers, Karl, E.: Software Requirements, Microsoft Press, 1999
11. Williams, R. C.; Walker, J. A.; Dorofee, A. J.: Putting Risk Management into Practice, Software Engineering Institute

PVM: Parallel View Maintenance under Concurrent Data Updates of Distributed Sources[⋆]

Xin Zhang, Lingli Ding, and Elke A. Rundensteiner

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

Abstract. Data warehouses (DW) are built by gathering information from distributed information sources (ISs) and integrating it into one customized repository. In recent years, work has begun to address the problem of view maintenance of DWs under concurrent data updates of different ISs. The SWEEP solution is one solution that does not require the ISs to be quiescence, as required by previous strategies, by employing a local compensation strategy. SWEEP however processes all update messages in a sequential manner. To optimize upon this sequential processing, we now propose a parallel view maintenance algorithm, called PVM, that incorporates all benefits of previous maintenance approaches while offering improved performance due to parallelism. We have identified two issues critical for supporting parallel view maintenance: (1) detecting maintenance-concurrent data updates in a parallel mode, and (2) correcting the problem that the DW commit order may not correspond to the DW update processing order due to parallel maintenance handling. In this work, we provide solutions to both issues. We have implemented both SWEEP and PVM in our EVE data warehousing system, and our studies confirm the multi-fold performance improvement of PVM over SWEEP.

1 Introduction

Background. Data warehousing [4] is an important technology to integrate data from distributed information sources (ISs) in order to provide data to for example decision support or data mining applications. Once a DW is established, the problem of maintaining it consistent with the underlying ISs under updates becomes a critical issue. It is typically more practical to maintain the DW incrementally [1,8,7] instead of recomputing the whole extent of the DW due to the

[⋆] 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 and Verizon Laboratories Incorporated for partial support of Xin Zhang.

large size of DWs and the enormous overhead associated with the DW loading process.

In recent years, there have been a number of algorithms proposed for view maintenance [8,9]. ECA [12] handles view maintenance under concurrent data updates of one centralized IS, while Strobe [13] and SWEEP [1] handle distributed ISs. There is a time period during which the extent of the DW will be temporarily inconsistent with that of the ISs.

Our Approach. In this paper, we propose an efficient algorithm called Parallel View Maintenance, or short PVM, that preserves all advantages of its predecessors, including SWEEP [1], while overcoming their main limitation in terms of performance caused by sequential processing. In particular, SWEEP handles data updates *sequentially* enforcing all data updates to queue at the DW manager until the data updates (DUs) received before them have been fully processed. Such sequential processing time in a DW system build over distributed ISs requires the system responsible for DW processing to wait for the processing of the DBMS server in each IS as well as for transmissions of results or update messages over the network. Although a variation called Nested-SWEEP handles multiple DUs in a more efficient way by reusing part of the query results, it puts more requirements on the DUs, e.g., requires the DUs to be non-interfering with each other in order to terminate. We will show that PVM more efficiently handles a set of concurrent data updates than previous solutions by parallelizing the DU handling process. Such optimization would lead to an improvement of the timeliness and hence quality of the DW content, which is important for time-critical applications like stock market and monitoring-control systems.

In this work, we identify two research issues central to enabling a parallel solution of view maintenance. The first issue is how to detect conflicting concurrent data updates within a parallel execution paradigm. The second issue is how to handle the random commit order of effects of data updates at the DW, which we call the out-of-order-DW-commit order, caused by the parallel execution of the view maintenance process without blocking incoming DUs. Our PVM solution proposed in this paper solves both problems.

A full implementation of PVM and the most popular contender in the literature (SWEEP) within our EVE [10] data warehousing testbed has been accomplished. Our evaluation based both on a cost model as well as on experimental studies has shown that PVM achieves a many-fold performance improvement over SWEEP [11].

2 Background on View Maintenance

View maintenance is concerned with maintaining the materialized views up-to-date when the underlying information sources (ISs) are modified over time. The straightforward approach of recomputing the whole view extent for each data update is not realistic. Instead, an incremental solution that only calculates the effect of each data update from the ISs and updates the data warehouse incrementally is typically more efficient. If two data updates are separated far enough in time so that they do not interfere with each other during maintenance, then this incremental approach is straightforward. In this case, when a data update ΔR_i happened at base relation R_i and is received at the middle layer, the incremental changes are computed by sending down subqueries of the following query [6,7] to the respective ISs:

$$\prod_{attributes} \sigma_{predicates}(R_1 \bowtie ... \bowtie \Delta R_i \bowtie ... \bowtie R_n) \tag{1}$$

We introduced the notations used in this paper in Table 1. The subqueries are generated in the sequence that starts from joining the relations on the left side of ΔR_i, and then continues to join the relations on the right side.

Table 1. Notations

Notation	**Meaning**
DU_i	A Data Update with unique subscript i.
Q_i	A Query for handling DU_i.
QR_i	The Query Result of Q_i.
IS_m	An Information Source with unique index m.
$SQ_{i,m}$	A Sub-Query of Q_i being sent to IS_m.
$SQR_{i,m}$	A Sub-Query Result of $SQ_{i,m}$.

However, due to the autonomous nature of the sources participating in the data warehouse, this kind of separation in time cannot always be guaranteed. Hence, we have to maintain the data warehouse under possibly concurrent data updates that are not separated far enough to accomplish the naive incremental maintenance illustrated above. Recent work by [13,1] proposes special techniques such as compensation queries to correct for possible errors caused by the concurrency.

3 PVM: A Parallel View Maintenance Solution

3.1 View Maintenance under Concurrent Data Updates

Because the data updates are happening concurrently at the ISs, one data update DU_2 at IS_j could interfere with the query processing of a query sent by the middle layer to this IS_j to handle another data update DU_1. We call this data update DU_2 a **maintenance-concurrent** data update as it takes place concurrently with the maintenance process servicing other updates.

Definition 1. *Let DU_i and DU_j denote two DUs on IS_m and IS_n respectively. The data update DU_j is called* **maintenance-concurrent** *with the update DU_i, denoted by $DU_j \vdash DU_i$, iff:*

i) DU_i is received earlier than DU_j at the middle layer, (i.e., $i < j$)

ii) DU_j is received at the DW **before** *the answer of the sub-query $SQ_{j,n}$ on the same IS_n generated by the DW for handling DU_i. An update DU_p is* **maintenance-concurrent**, *if there exists at least one DU_q for some $q < p$ such that $DU_p \vdash DU_q$.*

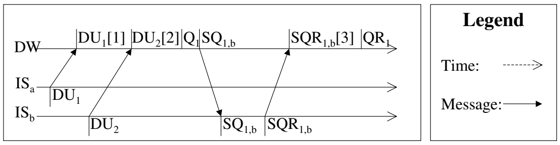


Fig. 1. Definition of a **maintenance-concurrent** Data Update

For example, in Figure 1, DU_1 occurs at IS_a and DU_2 occurs at IS_b respectively. We assume that DU_1 is received at time $t = 1$, denoted as $DU_1[1]$, which is earlier than DU_2 at time $t = 2$, denoted as $DU_2[2]$, at the DW middle-layer. Then the DW middle-layer generates a query Q_1 in order to handle $DU_1[1]$. The query Q_1 will then be broken into sub-queries to be handled by each IS. The sub-query $SQ_{1,b}$ that is generated from Q_1 is sent to IS_b. Since DU_2 occured before IS_b received $SQ_{1,b}$, DU_2 is received earlier than the query result of $SQ_{1,b}$, denoted as $SQR_{1,b}$, at the DW. Then, the DW will assign $t = 3$ to $SQR_{1,b}$ and denotes it as $SQR_{1,b}[3]$ in the middle layer. At the DW middle layer, based on the timestamp, we can see that $DU_1[1]$ was received earlier than $DU_2[2]$ and $DU_2[2]$ was received earlier than $SQR_{1,b}[3]$. This means that DU_2 affected the sub-query result $SQR_{1,b}$. By Definition 1, DU_2 is said to be a **maintenance-concurrent** data update.

3.2 Architecture of PVM

We now introduce the architecture of the PVM system in Figure 2. The *UpdateView* process is responsible for updating the materialized view of the DW and the *ViewChange* function is responsible for calculating the effect of a data update on the DW. *ViewChange* is the function with the largest overhead as it is (1) waiting for remote queries to be processed by ISs, (2) handling multiple queries, and (3) possibly restricted by processing these remote queries one by one in some sequential order. This results in waits due to IS processes and transmissions over the network. Hence, we propose to multi-thread this functionality to handle more than one data update at the same time.

The *AssignTimeStamp* process assigns a middle-layer timestamp to each incoming message, including both data updates and query results, to help with the detection of concurrent data updates. Every item in the UMQ now has an annotated timestamp, denoted by t in Figure 2. If one update and one query result are received at the same time, we know that they come from different ISs.

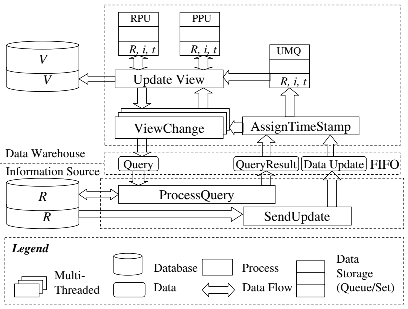


Fig. 2. Modules of PVM

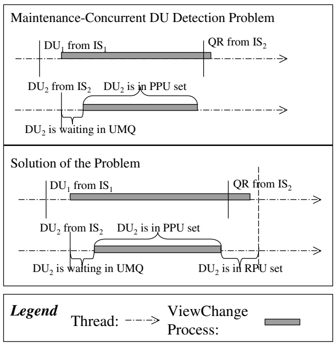


Fig. 3. Maintenance-concurrent DU Detection Problem

That means, the query result will not be affected by the data update. Hence, the order assigned to those two messages will not cause any additional concurrencies. Only the data updates in the UMQ received earlier than the query result and from the same IS as the query result will be treated as concurrent data updates.

In order to parallelize the view maintenance process, we introduce two modules, i.e., *Related Processed Update* (RPU) set and the *Parallel Processing Update* (PPU) set, where it keeps all the updates processed in parallel, that both are not found in other VM systems.

3.3 Issue 1: Detection of Maintenance-Concurrent DUs under Parallel Execution

A Motivating Example Illustrating the Problem. In Figure 3 there are two data updates DU_1 and DU_2 from ISs IS_1 and IS_2, respectively. DU_1 arrived at the DW before DU_2. So DU_1 is handled first. However, DU_2 affects the query result QR processed at IS_2, which is used to calculate the effect of DU_1. So DU_2 is a **maintenance-concurrent** data update by Definition 1. In Figure 3 we can see that DU_2 waited in the UMQ to be handled and then is stored in the PPU set, while being handled. They are being finally erased from the PVM system after having been handled. However, we assumed that DU_2 is handled faster than the processing of DU_1. So, before DU_1 receives the query result QR from the IS_2, DU_2 has already been removed from the PVM system. Hence, PVM can no longer detect the **maintenance-concurrent** data update DU_2 by only checking UMQ and PPU. Thus the final state of the DW after handling the DU_1 and DU_2 updates would be inconsistent with the state of the ISs.

Solution of Detecting Maintenance-Concurrent DU. The reason for this problem is that there is no tracking of data updates that have been completely handled already but may still be needed for the purpose of future concurrent data update detection. We address this problem by keeping the completely handled data updates in a special holding place where we can check for potential concurrent data updates.

Definition 2. [Related Processed Updates]*: A data update (DU) already completely handled by the middle layer is a* **related processed update** *if there exists at least one data update DU_j received before DU_i by DW that hasn't been completely handled by the ViewChange process yet.*

In order to solve this problem, we put a data storage into the PVM system (Figure 2) in addition to the PPU set, called the RPU set. It will maintain all related processed updates until their retention becomes no longer necessary. The *ViewChange* process handles one data update DU_i and commits the effect of that DU_i to the data warehouse, it will first clean up the RPU to remove all the unrelated processed DUs that do not satisfy Definition 2. This can be done by simply checking the timestamps and IS indices of DUs in PPU and RPU. Then it will check whether the DU_i is a RPU. If it is a RPU, the *ViewChange* process will put it into the RPU set. Figure 3 illustrates the relative time intervals for the DU along the time-line of *ViewChange* process.

3.4 Issue 2: Out-of-Order DW Commit

A Motivating Example. In a parallel environment, view maintenance algorithms may work on multiple data updates. Some of the DUs may need a shorter processing time than others. For example, different DUs terminate at different iterations due to an empty query result or due to different network delays on connections to different IS. If every view maintenance thread could commit their change to the data warehouse independently, this could allow us to avoid such unnecessary waits and thus to increase system performance. However, in that case, the commit order of the updates will not match the order of the data updates received at the middle-layer. Hence, it will result in an inconsistent state of the DW extent, meaning that there is no IS space state that matches it.

Assume we have two relations A and B with DW defined by $DW = A \times B$. The extents of A, B and DW are shown in Figure 4. Two data updates DU_1 and DU_2 happened on B and A respectively, namely DU_1 adds $< 3 >$ to B, while DU_2 deletes $< 1 >$ from A. The middle layer received first DU_1 and then DU_2. The effect of DU_1, denoted by ΔDW_1 defined by $A \times DU_1$, is shown in Figure 4. The effect of DU_2, denoted by ΔDW_2, is calculated after receiving DU_2. It is defined by $DU_2 \times B'$ as shown in Figure 4. After we apply the two Δ DW-s in the order of ΔDW_1 and ΔDW_2 as described in *Correct Commit Order* part of Figure 5, we get the correct DW state.

If we reverse the commit order of the two Δ DW-s (and the original DW can only count positive tuples as is the state-of-the-art for current DBMS systems), we update the DW in the wrong commit order as shown at the bottom of Figure 5. First, we would subtract ΔDW_2, which given that $< 1, 3 >$ does not yet exist in DW, would be equal to a no-op. Then after union ΔDW_1 with the DW, we get the wrong DW extent depicted in Figure 5, which has a faulty tuple $< 1, 3 >$ with counter 1. The correct answer instead should have been the DW extent given in the *Correct Commit Order* part of Figure 5.

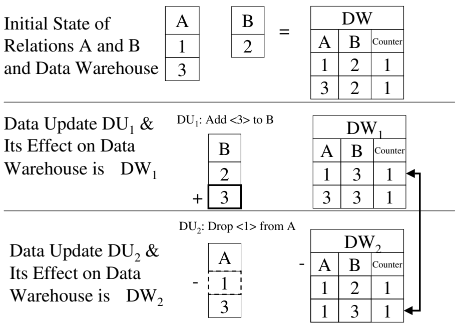

Initial State of Relations A and B and Data Warehouse
A
1
3
B
2
=
DW
A B Counter
1 2 1
3 2 1
Data Update DU1 & Its Effect on Data Warehouse is DW1
DU1: Add <3> to B
DW1
1 3 1
3 3 1
Data Update DU2 & Its Effect on Data Warehouse is DW2
DU2: Drop <1> from A
DW2
1 2 1
1 3 1

Fig. 4. Environment of Example

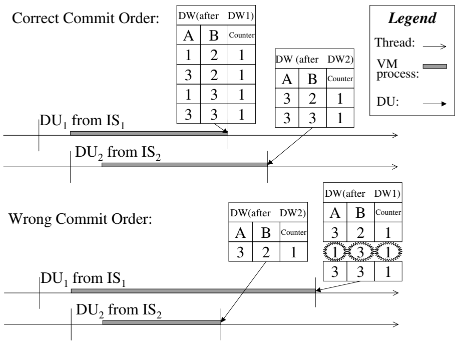


Fig. 5. Example of **Out-of-Order-DW-Commit** Problem

Problem Definition. If we are doing parallel incremental view maintenance of the DW, the order of committing the effects of the two data updates may result in different extents of DW even though only one of them can be the correct one. **DW-Commit-Order** is the order in which data update effects are committed to the DW. The problem of inconsistency between the final state of DW and the IS space caused by the variant **DW-Commit-Order** is called the **Out-of-Order-DW-Commit** problem.

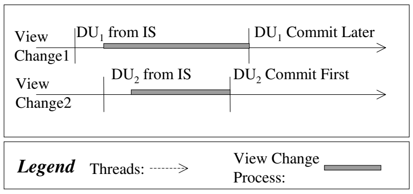


Fig. 6. Out-of-Order-DW-Commit Order Problem

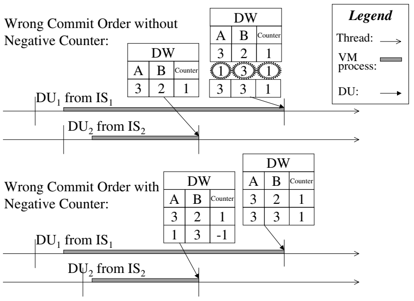

Wrong Commit Order without Negative Counter:
DW
DU1 from IS1
DU2 from IS2
Wrong Commit Order with Negative Counter:
1 3 -1
Legend
Thread:
VM process:
DU:

Fig. 7. Example of Solution of Out-of-Order DW Commit Order Problem

Theorem 1. *The* **Out-of-Order-DW-Commit** *problem will only occur when first an add-tuple and then a delete-tuple, which both modify the same tuples in the DW, are received by the middle layer and both are handled in parallel by PVM.*

The proof can be found in our technical report [11]. Figure 6 illustrates the **Out-of-Order-DW-Commit** problem with a time-line view. This problem may happen when the later handled data update DU_2 is handled much faster than the previously handled data update DU_1. In this case the DU_2 will complete first and commit its effect to the DW prior to the effect of DU_1 being committed to the DW.

Negative-Counter Solution. We now propose a technique called the negative-counter solution that guarantees that the extent of the DW after the parallel incremental view maintenance of multiple DUs will be correct. The basic principle underlying the solution is that we store the DW extent as unique tuples with a counter that shows how many duplicates exist of each tuple. For example, <1,3>[4] means four tuples with values <1,3> exist. The unique twist here is that we permit the counter to be **negative**. Then, for adding (deleting) one tuple, if the tuple already exists in the DW, we increase (decrease) the counter by one, else we create a new tuple with counter '1' ('-1'). We remove a tuple whenever its counter reaches '0'. When the user accesses the DW, any tuple with a counter ≤ 0 will be not visible.

Figure 7 compares the extent of the DW from our previous example from Section 3.4 with and without the negative counter. In the upper half of the figure, DW keeps only tuples with positive counters at any time. After the effect of DU_2, ΔDW_2, is committed, we removed $< 1, 2 >$ but the delete-tuple update $< 1, 3 >$ is invalid since $< 1, 3 >$ is not in the DW yet. Then, after we commit the ΔDW_1 later, we now have added the faulty tuple $< 1, 3 >$ into the DW, which should be deleted (however, the deletion was attempted too early).

In the lower half of Figure 7, the DW can keep tuples with a negative counter. Then the addition of that faulty tuple in the future will remove the tuple with the negative counter from the DW. So, the tuple with a negative counter effectively remembers what tuple should be deleted but cannot be deleted at the current time due to it not yet being in the DW. For example, in the lower part, we can see that DW keeps the tuple $< 1, 3 >$ with negative counter **-1** and thus effectively remembers that the $< 1, 3 >$ should be deleted eventually. After committing the ΔDW_1, the tuple $< 1, 3 > [-1]$ is compensated with the tuple $< 1, 3 > [1]$ in ΔDW_1 that given $< 1, 3 > [-1]+ < 1, 3 > [1] =< 1, 3 > [0]$, the tuple is finally completely removed from DW. Hence the final state of DW is consistent with the state of the ISs. The correctness proof is in [11].

3.5 Consistency Level

PVM in its purest form only ensures the *Convergence* level of consistency [12] in order to allow for maximal parallelism of the maintenance process of each individual update. However, if it became important to not only achieve *Convergence* but also *Complete Consistency*, we can enforce that the effects of data updates are committed in the order as they arrive at the DW by adding an extra buffer in the *Update View* module of Figure 2. This buffer would keep the ready-to-commit updates in the order that they were received, and the *Update View* module will only update the view when the head of the buffer is filled.

4 Related Work

View maintenance methods concentrate instead on ensuring consistency of the DW when the materialized views are not self-maintainable. The ECA [12] family of algorithms introduces the maintenance problem and solves it partially, i.e., for one single IS. Strobe [13] and SWEEP [1] are two view maintenance algorithms for multiple ISs both of which focus on the concurrency of data updates.

The Strobe [13] algorithm introduces the concept of queueing the view updates at the DW and committing the updates to the DW only when the unanswered query set (UQS) is empty. The algorithm solves the consistency problem but is subject to the potential threat of infinite waiting, i.e., the DW extent may never get updated. Other mechanisms are based on requiring the ISs to timestamp the view updates [2], i.e., some global timing synchronization of the environment. So far, most consistency maintenance methods either have the infinite waiting deficiency or need a global timestamp service. Like SWEEP, PVM doesn't have any of the above limitations to handle multiple data updates.

The SWEEP [1] family of algorithms however succeeded in eliminating the above mentioned limitations by applying local compensation techniques. SWEEP uses special detection methods for concurrent data updates that don't need a global time stamp mechanism. It also requires no quiescent state before being able to update the DW. Beyond this, PVM now addresses the bottleneck of SWEEP's sequential processing by exploiting the inherent parallelism of this view maintenance process due to the distributed ISs each with their independent servers.

Nested-SWEEP [1] is used to handle a set of DUs by reusing the query results. Due to its recursive solution, it requires non-interference of the DUs, otherwise, an infinite recursive call may result in a maintenance failure. PVM doesn't apply any recursive process optimization to share the query result as in Nested-SWEEP, rather it instead parallelizes the SWEEP maintenance process.

5 Conclusions

In this paper, we have investigated the problem of parallel view maintenance. First, we identified the bottleneck in terms of sequential handling of DUs by the current state-of-the-art VM solution called SWEEP [1]. The SWEEP algorithm has the least limitations compared to alternate contenders in the literature, but it adopts sequential processing when handling a set of data updates. In this work, we have identified several open issues to achieve parallel view maintenance, notably, concurrent data update detection in a parallel execution mode, and the out-of-order-DW-commit problem. We present an integrated solution PVM for handling these problems. We have implemented both SWEEP and PVM in our EVE data warehousing system [3]. The experimental results demonstrate that PVM achieves a multi-fold performance improvement compared to SWEEP, however, due to space limitations these results cannot be shown here, but in our technical report [11].

References

1. D. Agrawal, A. El Abbadi, A. Singh, and T. Yurek. Efficient View Maintenance at Data Warehouses. In *Proceedings of SIGMOD*, pages 417–427, 1997.
2. E. Baralis, S. Ceri, and S.Paraboschi. Conservative TimeStamp Revised for Materialized View Maintenance in a Data Warehouse. In *Workshop on Materialized Views*, pages 1–9, 1996.
3. J. Chen, X. Zhang, S. Chen, A. Koeller, and E. A. Rundensteiner. DyDa: Data Warehouse Maintenance in Fully Concurrent Environments. In *Proceedings of SIGMOD'01 Demo Session*, May 2001.
4. 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.
5. A. Gupta, H.V. Jagadish, and I.S. Mumick. Maintenance and Self-Maintenance of Outer-Join Views. In *Next Generation Information Technologies and Systems,* 1997.
6. 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.
7. A. Gupta, I. S. Mumick, and V. S. Subrahmanian. Maintaining Views Incrementally. In *Proceedings of SIGMOD*, pages 157–166, 1993.
8. M. K. Mohania, S. Konomi, and Y. Kambayashi. Incremental Maintenance of Materialized Views. In *Database and Expert Systems Applications (DEXA)*, pages 551–560, 1997.
9. A. Nica and E. A. Rundensteiner. View Maintenance after View Synchronization. In *International Database Engineering and Applications Symposium (IDEAS'99)*, pages 213–215, August, Montreal, Canada 1999.
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 1999.
11. 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, Computer Science Department, May 1999.
12. 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.
13. 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.

An Experimental Performance Evaluation of Incremental Materialized View Maintenance in Object Databases

M. Akhtar Ali[1], Norman W. Paton[2], and Alvaro A.A. Fernandes[2]

[1] School of Computing and Mathematics, University of Northumbria
Newcastle Newcastle Upon Tyne NE1 8ST, UK
akhtar.ali@unn.ac.uk

[2] Department of Computer Science, University of Manchester,
Oxford Road, Manchester M13 9PL, UK.
{norm,alvaro}@cs.man.ac.uk

Abstract. The development of techniques for supporting incremental maintenance of materialized views has been an active research area for over twenty years. However, although there has been much research on methods and algorithms, there are surprisingly few systematic studies on the performance of different approaches. As a result, understanding of the circumstances in which materialized views are beneficial (or not) can be seen to lag behind research on incremental maintenance techniques. This paper presents the results of an experimental performance analysis carried out in a system that incrementally maintains OQL views in an ODMG compliant object database. The results indicate how the effectiveness of incremental maintenance is affected by issues such as database size, and the complexity and selectivity of views.

1 Introduction

Incremental view maintenance is the propagation to a materialised view of updates to base data that affect the value of the view. Proposals for solutions to the incremental view maintenance (IVM) problem are numerous [7], and individual solutions differ in the data model, view language, data to which access is required, and timeliness of update propagation. The variety of different techniques available and the intrinsic complexity of the trade-offs involved in deciding when to materialize has promoted the selection of views for materialization to a research topic in its own right [4,10]. However, although materialized view selection is crucially dependent on an understanding of the cost issues associated with incremental maintenance, there has been surprisingly little work seeking systematically to evaluate the performance of proposals for IVM systems.

This paper presents the results of an experimental performance evaluation of the MOVIE (Materialized Object-oriented VIEws) system [1], which provides an algebraic solution to the incremental maintenance of OQL [3] views. The performance evaluation has made use of the OO7 [2] schema and data generation procedure. The paper is structured as follows. Section 2 provides an overview of the MOVIE system including some of the technical background that underpins it. Section 3 describes the schema, queries, views and updates used in the experiments. Section 4 presents the experiments conducted in the performance evaluation. Finally, Section 5 concludes and points to future work.

2 Incremental Maintenance of OQL Views

2.1 Technical Background

The ODMG standard for object databases includes an object model (OM), an object definition language (ODL) and an object query language (OQL). Familiarity with the ODMG model and OQL is assumed to the level that is required to interpret schema diagrams and OQL queries.

The λ-DB system [6] implements a monoid algebra over the ODMG object model (not including arrays and dictionaries). OQL queries in λ-DB are translated into algebraic expressions that are optimized before they are mapped to physical execution plans. An execution plan is then translated into C++ and compiled. The resulting executable, when run, evaluates the original OQL query.

2.2 An Overview of MOVIE

The MOVIE system is an implementation of our solution to the IVM problem for OQL views, and is described more fully in [1]. The solution assumes the availability of the update event, the changes made to the database (hereafter referred to as the *delta)*, the MV definition, and the current materialized state of the view. In order to achieve incremental maintainability for all update operations, we also assume the availability of references to the base objects that contribute data to the MV, and of the base extents required for materializing it. Our solution works for MVs that refer to any ODMG type (excluding array and dictionary) and that are definable using the reduce, join, get, nest and unnest bulk operators of the algebra proposed in [6] (excluding self joins). It is valid for any update operation in the ODMG standard (e.g., `new()` and `delete()` on objects, `insert_element()` on collections, etc.). In terms of practicality, our solution yields incremental maintenance plans (IMPs) at the algebraic level and, to the best of our knowledge, for object databases, is the first one to do so. This makes it easier to integrate our solution into the kind of query processing frameworks that mainstream database management systems (DBMSs) rely on (e.g. [5]).

When an MV is defined, its definition is traversed to identify the kinds of update events that might require propagation of changes to the MV. For each kind of event, an algebraic IMP is constructed that can compute the required changes to the MV. The core of our solution is, therefore, the generation of an IMP that is appropriate for each kind of update event. In our approach, two kinds of IMP (described below) suffice to compute the changes required in the MV as a result of any update operation in the ODMG standard. Immediately after an update event takes place which implies the need to update an MV, the corresponding delta (comprising the old and the new state of affected objects) is made available and the associated IMP (which uses the delta) is evaluated to compute the changes needed.

The comprehensive nature of our solution with respect to update operations requires that the OIDs of objects that contribute data to the MV are also materialized. This is achieved by generating from an MV definition `v` another view definition, which we refer to as *OIDs_for_*`v`, which is itself compiled, evaluated and materialized too. Thus, the OIDs of objects that contribute data for an instance of `v` are associated with the OID of that instance in *OIDs_for_*`v`. This space overhead (which naturally induces some time overhead) is present in our solution for it to be exhaustive over the set of operations in the ODMG standard.

Different forms of IMPs are generated depending on the event type and the category of the view[1], as follows. We note that similar categories of plan are present in other IVM systems, and thus that lessons learned from the experiments described below are potentially relevant to other proposals.

1. **Planting a delta** – For some kinds of updates (e.g. `onInsertTo`), the constructed IMP computes the changes required to `v` by evaluating the view over the delta to the affected base extent, rather than over the base extent itself, while accessing all other base extents referenced in the MV. As well as generating the additional data for the view, the incremental plan also projects the OIDs of the objects represented in the view.
2. **Joining a delta with materialized OIDs** – For other kinds of updates (e.g. `onDeleteTo`), the constructed IMP joins *OIDs_for_*`v` with the delta in order to identify MV objects that are affected by the update. The idea is to avoid access to base extents whenever possible. This only applies to ReduceGetJoin views.

3 Experimental Views, Queries, and Updates

Databases: The databases used in our experiments are based on the OO7 Benchmark [2]. We did not use the three OO7 database size configurations, but rather generated databases of different sizes by varying the number of modules (the top level class in OO7). The actual parameters given to the OO7 data generation program, plus some information on the resulting database, are given in Figure 1 (where the database sizes correspond to four, eight, twelve, sixteen and twenty times the size of the small database in the OO7 benchmark).

Testbed: The experiments reported in this paper were carried out on a PC with the following hardware and software: Intel Pentium Pro processor, 200 MHz, 256KB cache, 128MB RAM, 4GB SCSI Hard Disk (where the system software and 128MB of swap space reside); RedHat Linux 6.0 Kernel 2.2.5-22, SHORE 1.1.1 and λ-DB 0.5.

[1] A complete list of event types and view categories, the latter characterised by the algebraic operators that are used to evaluate the view, is given in [1].

	Database Size				
	db1	db2	db3	db4	db5
fan-out	6	6	6	6	6
modules	4	8	12	16	20
objects	288,380	576,760	865,140	1,153,520	1,441,900
size (MB)	45	90	133	177	222

Fig. 1. Databases Used in the Evaluation

Views: The views we use are given in Figure 2. Note that for each MV v, we denote the corresponding non-materialized view by $vVir$. For example, to the view `complexView1` (in Figure 2.b) there corresponds a view named `complexView1Vir` that is otherwise identical to `complexView1` but is not materialized. We have defined the views in Figure 2 for the purpose of investigating the following hypotheses:

1. The selectivity of the view influences the performance of incremental view materialization.
2. The structural complexity of the view influences the performance of incremental view materialization.
3. The benefits of incremental view materialization increase with an increase in database size.

Selectivity: Different view selectivities (defined as the ratio of the number of objects selected by a query to the number of input objects) arise as a result of a view containing predicates that filter (to different degrees) the input data.

In the experiments, a template view (shown in Figure 2.a) is defined that retrieves the `id` and `type` of `compositePart` objects using different predicates to filter the input. A set of five views with different selectivities is derived by instantiating the template, as follows. Each view differs in the predicate ϕ_i in the `where` clause. Each predicate is obtained by instantiating the template ϕ_i with `c.id` $<=k_i$ where $k_1 = 1200$ to $k_5 = 6000$ in increments of 1200, thereby yielding five different views with selectivities from 0.2 to 1.0 in increments of 0.2. In experiments involving selectivity, the database for which the view is defined is `db3`. In this database, there are 6000 `compositePart` objects. The `id` of these objects is set sequentially from 1 to 6000, so it is possible to control the selectivity of the view.

Complexity: With regard to the structural complexity of the view, we take it to vary with the number and the kind of algebraic operators needed to evaluate the query part of the view. Intuitively, a query that contains two joins is more complex than a query that contains one join. Therefore, we define four views (in Figures 2.b to 2.e) with increasing complexity in order to perform experiments over the materialized and virtual versions of the views.

Database Size: With regard to database size, the number of modules varies uniformly across the five databases. We are interested in finding the impact of

```
define viewSel_i() as
select struct(
              ComPartId:c.id,
              CType:c.type)
from   c in CompositeParts
where  φ_i
```

(a) Selectivity Dependent (on ϕ_i) Views

```
define complexView1() as
select struct(
              ComPartId:c.id,
              CType:c.type)
from   c in CompositeParts
where  c.id <= 200
```

(b) `complexView1`: A Reduce-Get View

```
define complexView2() as
select struct(
              ComPartId:c.id,
              CType:c.type,
              AtomPartId:a.id)
from   c in CompositeParts,
       a in c.parts
where  c.id <= 200 and
       c.buildDate > a.buildDate
```

(c) `complexView2` = `complexView1` plus Unnest

```
define complexView3() as
select struct(
              ComPartId:c.id,
              CType:c.type,
              AtomPartId:a.id,
              BaseAssmId:b.id)
from   c in CompositeParts,
       a in c.parts,
       b in c.usedInPriv
where  c.id<=200 and
       c.buildDate>a.buildDate and
       c.buildDate>b.buildDate
```

(d) `complexView3` = `complexView2` plus Unnest

```
define complexView4() as
select struct(
              ComPartId:c.id,
              CType:c.type,
              AtomPartId:a.id,
              BaseAssmId:b.id,
              DocId:d.id)
from   c in CompositeParts,
       a in c.parts,
       b in c.usedInPriv,
       d in Documents
where  c.id<=200 and
       c.buildDate>a.buildDate and
       c.buildDate>b.buildDate and
       c.documentation=d
```

(e) `complexView4` = `complexView3` plus Join

```
define dbSizeView() as
select struct(
              ComPartId:c.id,
              CType:c.type,
              DocId:d.id)
from   c in CompositeParts,
       d in Documents
where  c.documentation = d
```

(f) `dbSizeView`: A Size-Sensitive View

Fig. 2. View Templates and Views Used in the Evaluation

```
Update:delete a compositePart object
Event: onDeleteTo CompositeParts
```

(a) U1: A Deletion Update Event

```
Update:assign "type111" to the type
       attribute of the compositePart
       object with id = 500
Event: onModifyAttr compositePart.type
```

(b) U2: An Attribute Modification Event

```
Update:insert a new object into
       the Documents extent
Event: onInsertTo Documents
```

(c) U3: An Insertion Update Event

Fig. 3. Update Events Used in the Evaluation

database size on the performance of incremental view materialization. A size-sensitive view, `dbSizeView` (in Figure 2.e), provides a means of showing the impact of database size on the performance of materialized views. The number of `compositePart` objects per `module` is 500 and the number of `document` objects is the same. In `db1`, there are 2000 `compositePart` and `document` objects. In contrast, in `db5`, there are 10000 `compositePart` and `document` objects. The cardinality of the result of `dbSizeView` increases from 2000 in the case of `db1` to 10000 in the case of `db5`.

Queries: The queries used in experiments for contrasting the time efficiency of answering queries against materialized and against virtual views are simple select statements over the derived or materialized views. In Section 4, these are referred to as MQ_i and VQ_i, $1 \leq i \leq 6$, when they are evaluated over the materialized and the virtual views, respectively.

Updates: The updates that are used in experiments involving propagation are given in Figure 3.

To exemplify the two kinds of IMP that are generated by MOVIE, consider the `dbSizeView` in Figure 2.f. For the update event U2 in Figure 3.b to be incrementally propagated, changes are computed by joining a delta with materialized OIDs. This results in the algebraic query plan shown in Figure 4.a. In contrast, for the update event U3 in Figure 3.c to be incrementally propagated, changes are computed by planting a delta. This results in the algebraic query plan shown in Figure 4.b. The nature of these plans is discussed in more detail in [1], and the mapping from OQL to the object algebra is described in [6].

```
reduce(set,
     join(set,
          get(set, $\Delta_{\text{onModifyAttr compositePart.type}}$,$\delta$,and()),
          get(set,OIDs_for_dbSizeView,mat_oids,and()),
          and(eq(mat_oids.DO, $\delta$)),
          none),
     X1,
     struct(mat_obj=mat_oids.VO, CType=$\delta$.type),
     and())
                                        (a)
reduce(set,
     join(set,
          get(set, $\Delta_{\text{onInsertTo Documents}}$,d,and()),
          get(set,CompositeParts,c,and()),
          and(eq(c.documentation, d)),
          none),
     X1,
     struct(ComPartId=c.id), CType=c.type, DocId=d.id),
     and())
                                        (b)
```

Fig. 4. Computing Changes to `dbSizeView` Following (a) U2 and (b) U3

4 Experiments and Discussion

The performance of incremental view materialization can be evaluated by considering:

1. The cost of query answering over the MV and over its virtual equivalent.
2. The cost of incrementally maintaining the MV by update propagation.
3. The cost of incrementally maintaining the MV in comparison with the cost of answering a query over its virtual equivalent.

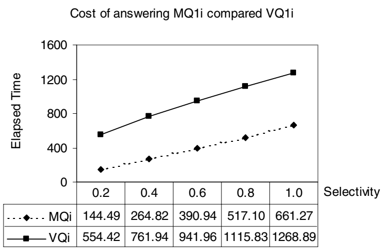
Three experiments are designed to test each of the hypotheses in Section 3 exploring the above three kinds of cost. The experiments reported were run in cold states so that one experiment is not favoured against another due to object caching. After each individual experiment, the database server was shut down and restarted for the next experiment, and the linux cache was flushed. Each individual experiment was run three times and the average taken. The values reported for relative cost are rounded to the nearest integer value.

4.1 Experiment 1: Varying Selectivity

This experiment explores the impact of view predicate selectivity on the performance of IVM. We break the experiment into parts, each corresponding to one of the hypotheses under scrutiny.

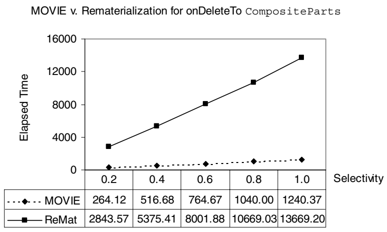
Query Answering: MQ1i and VQ1i. The elapsed times taken to answer MQ1i and VQ1i, $1 \leq i \leq 5$, using the predicates defined with `viewSel_`i are measured and compared. The results are plotted in Figure 5.a.

They reveal that the cost of answering both MQ1i and VQ1i increases with an increase in selectivity. However, the relative cost of evaluating VQ1i compared with MQ1i reduces with increases in selectivity. This is consistent with expectations, because evaluating VQ1i always requires scanning of the complete extent of `CompositeParts`, whereas the number of objects scanned by MQ1i increases with selectivity. With selectivity 1.0, answering the query over the MQ1i is less costly than over the VQ1i because, although the queries retrieve the same number of objects, the view objects are smaller than `compositePart` objects.



Selectivity	0.2	0.4	0.6	0.8	1.0
◆ MQi	144.49	264.82	390.94	517.10	661.27
■ VQi	554.42	761.94	941.96	1115.83	1268.89

(a) Query Answering



Selectivity	0.2	0.4	0.6	0.8	1.0
◆ MOVIE	264.12	516.68	764.67	1040.00	1240.37
■ ReMat	2843.57	5375.41	8001.88	10669.03	13669.20

(b) View Maintenance

Fig. 5. Results of Experiment 1: Varying Selectivity

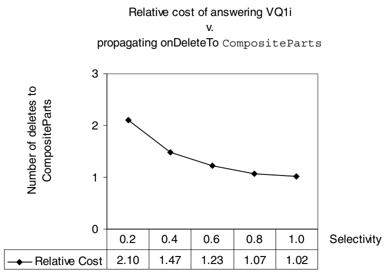
View Maintenance: Propagating U1 to `viewSel_`*i.* We measure the elapsed time taken to propagate update U1 (in Figure 3.a) – which requires maintenance of `viewSel_`i – using MOVIE and using rematerialization. The results are plotted in Figure 5.b.

They reveal that the cost of view maintenance increases with an increase in selectivity in both cases but that MOVIE outperforms rematerizalization by around ten times in general. This suggests that, although the maintenance costs increase with an increase in selectivity, IVM can be advantageous independently of view selectivity.

Relative Cost (Query/Update): Answering VQ1i v. Propagating U1. This part of the experiment tries to identify the conditions under which incrementally maintaining MVs is beneficial. We do so by investigating how many updates can be propagated incrementally to an MV in the time taken to answer the corresponding virtual query. We calculate the ratio of the elapsed time taken to answer VQ1i to that taken to propagate update U1 (in Figure 3.a) to `viewSel_`i. The results are plotted in Figure 6.a.

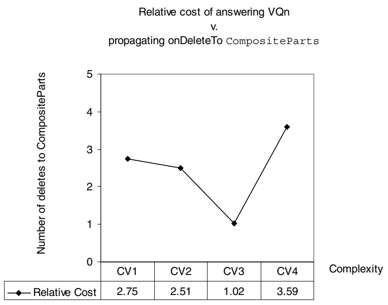
They reveal that, for the view with selectivity of 0.2, two updates can be propagated incrementally for the cost of evaluating a query against the corresponding virtual view. With an increase in the selectivity that ratio reduces to one, suggesting that, for views with high selectivity, if updates are at all frequent, incremental maintenance may not be beneficial.

In summary, the benefits of IVM seem to be greater when views have lower selectivities. This result conforms with general expectations, in that the materialized views with low selectivities involve smaller data sets than those associated with the corresponding virtual views. Space has prevented the inclusion of selectivity experiments on more complex views.



	0.2	0.4	0.6	0.8	1.0
Relative Cost	2.10	1.47	1.23	1.07	1.02

(a) Relative Cost For Expt 1



	CV1	CV2	CV3	CV4
Relative Cost	2.75	2.51	1.02	3.59

(b) Relative Cost For Expt 2

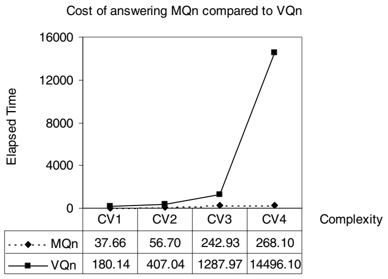
Fig. 6. Relative Query and Update Costs in Experiments 1 and 2

4.2 Experiment 2: Varying Structural Complexity

This experiment seeks to show how the complexity of the views may affect the performance of IVM. Again, we break the experiment into parts, each one corresponding to one of the hypotheses under scrutiny. In experiments involving complexity, the database used is db3.

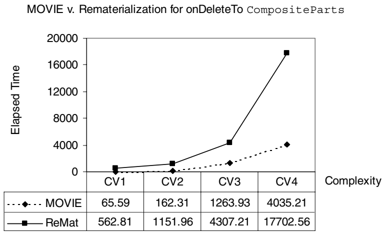
Query Answering MQn and VQn. The elapsed times taken to answer MQn and VQn, $2 \leq n \leq 5$ are measured and compared. The results are plotted in Figure 7.a.

They reveal that, in the case of MVs, because the cardinalities increase from complexView1 to complexView3, the cost of answering queries over the MV increases significantly. However, since the cardinality of complexView3 is the same as that of complexView4 that cost increases by very little. This indicates that for access to MVs the dominant factor is cardinality. In the case of the virtual views, from complexView1 to complexView3 the cost increases but not very much as a consequence of the relatively low cost of unnesting. From complexView3 to



	CV1	CV2	CV3	CV4
◆ MQn	37.66	56.70	242.93	268.10
■ VQn	180.14	407.04	1287.97	14496.10

(a) Query Answering



	CV1	CV2	CV3	CV4
◆ MOVIE	65.59	162.31	1263.93	4035.21
■ ReMat	562.81	1151.96	4307.21	17702.56

(b) View Maintenance

Fig. 7. Results of Experiment 2: Varying Structural Complexity

`complexView4` there is a jump in the cost because `complexView4` requires a join. This jump may be partly the result of the expensive algorithm (viz., Block Nested Loop) that λ-DB selects to join `CompositeParts` and `Documents`. If a more efficient algorithm were used, the jump would likely be less pronounced.

In any case, the cost of answering queries over MVs increases with an increase in cardinality (rather than complexity), and thus the relative performance of MVs compared with their virtual counterparts depends very much on the selectivity of the view. Views with low selectivity are likely to be promising candidates for materialisation.

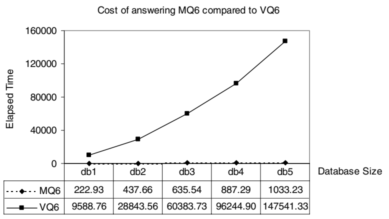
View Maintenance: Propagating U1 to `complexView`$_i$*.* We measure the elapsed time taken to propagate update U1 (in Figure 3.a) using MOVIE and using rematerialization. The results are plotted in Figure 7.b.

Update U1 (i.e., deletion of an instance of `compositeParts)` was chosen because it affects all of `complexView1` to `complexView4`. The results reveal that, for `complexView1` and `complexView2`, incremental maintenance is 9 and 7 times faster than rematerialization. However, for `complexView3`, MOVIE is only 3 times faster. The decrease is due to the fact that U1 requires the deletion of 100 objects from `complexView3` (as against 20 for `complexView2` and 1 for `complexView1`).

The elapsed time of rematerializing `complexView4` is 4 times more than that for `complexView3`. Similarly, MOVIE takes almost three times as long to propagate changes to `complexView4` as it takes in the case of `complexView3`. The bottom line is that the more complex the view the more expensive it is to maintain, no matter how it is being maintained.

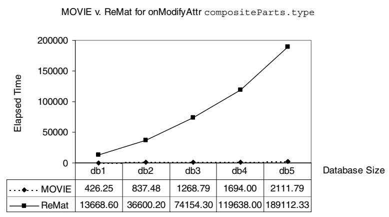
Relative Cost (Query/Update): Answering VQn, $2 \leq n \leq 5$*, v. Propagating U1.* This part of the experiment tries to identify the conditions under which incrementally maintaining MVs is beneficial by investigating how many updates can be propagated incrementally to an MV in the time it takes to answer the corresponding virtual query. In this experiment, we take the ratio of the elapsed time taken to answer `complexViewVir1` to `complexViewVir4` to that taken to propagate update U1 (in Figure 3.a) to all of `complexView1` to `complexView4`. The results are plotted in Figure 6.b. The results show that only quite a small number of updates (1 to 4) can be propagated in the time taken to evaluate the virtual view in this experiment. This can be seen as quite a disappointing result for incremental maintenance. The shape of the graph in Figure 6.b should not be considered to be of general significance. The “outlier” appears to be the propagation of U1 to `complexView3`. U1 can only be propagated to `complexView3` once in the time it takes to evaluate the virtual view. This is principally because of the high cost of incrementally propagating U1 to `complexView3`, which in turn is because a single occurrence of U1 causes 100 objects to be deleted from the view.

In summary, the experiment has not demonstrated any conclusive pattern relating to increased view complexity. The most striking result, from Figure 7.a, is the importance of the cardinality of the result to the incremental maintenance process.



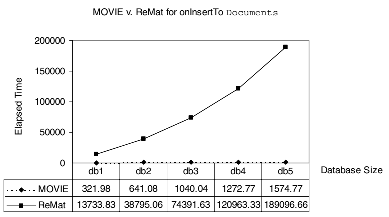
	db1	db2	db3	db4	db5
MQ6	222.93	437.66	635.54	887.29	1033.23
VQ6	9588.76	28843.56	60383.73	96244.90	147541.33

(a) Query Answering



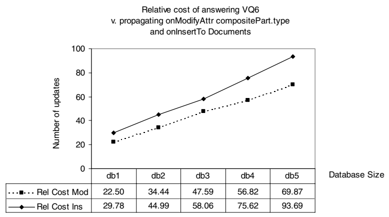
	db1	db2	db3	db4	db5
MOVIE	426.25	837.48	1268.79	1694.00	2111.79
ReMat	13668.60	36600.20	74154.30	119638.00	189112.33

(b) View Maintenance (1)



	db1	db2	db3	db4	db5
MOVIE	321.98	641.08	1040.04	1272.77	1574.77
ReMat	13733.83	38795.06	74391.63	120963.33	189096.66

(c) View Maintenance (2)



	db1	db2	db3	db4	db5
Rel Cost Mod	22.50	34.44	47.59	56.82	69.87
Rel Cost Ins	29.78	44.99	58.06	75.62	93.69

(d) Relative Cost (Query/Update)

Fig. 8. Results of Experiment 3: Varying Database Size

4.3 Experiment 3: Varying Database Size

This experiment explores the impact of database size on the performance of IVM. Once again, we break the experiment into parts that correspond to the hypotheses we are testing.

Query Answering: MQ6 and VQ6. Here the elapsed times taken to answer MQ6 and VQ6 are measured and compared. The results are plotted in Figure 8.a.

In the case of MQ6, the elapsed time taken to answer `dbSizeView` increases only slightly with an increase in the database size, reflecting the increase in the size of the materialized view. The benefit of answering queries over the MV rather than over the virtual counterpart increases as the database increases in size. For example, for the smallest database, it is 43 times faster to use the MV, whereas for the largest database, it is 143 times faster, because the join cost grows non-linearly.

View Maintenance: Propagating U2 and U3 to `dbSizeView`. We measure the elapsed times taken to propagate two different updates: U2 (in Figure 3.b) and U3 (in Figure 3.c), which require maintenance of `dbSizeView`, using MOVIE and using rematerialization. The results are plotted in Figures 8.b and 8.c.

Figure 8.b shows MOVIE outperforming rematerialization for an update event that modifies an attribute. For update U2, the IMP is generated by joining the delta with the materialized OIDs, i.e., no base data is accessed. The relative performance of MOVIE increases with an increase in the database size because costly joins with base data are avoided altogether.

Figure 8.c does a similar comparison for an update (viz., U3, which is an insert into a class extent) for which the IMP is generated by planting a delta, i.e., accessing base data. Although planting a delta is generally felt to be more expensive than joining with the materialized OIDs, MOVIE still performs better than rematerizalization. In fact, comparing the times in Figure 8.b and Figure 8.c, planting the delta works out faster than joining with the materialized OIDs in this case. Once again, the benefits of MOVIE increase with an increase in the database size. For example, in the smallest database, it performs 43 times better, whereas in the largest, it is 120 times faster than rematerizalization.

Relative Cost (Query/Update): Answering VQ6 v. Propagating U2 and U3. This part of the experiment tries to identify the conditions under which incrementally maintaining MVs is beneficial. We do so by investigating how many updates can be propagated incrementally to an MV in the time taken to answer the corresponding virtual query. We calculate the ratio of the elapsed time taken to answer VQ6 to that taken to propagate updates U2 and U3 (in Figure 3.b and 3.c, respectively) to `dbSizeView`. The results are plotted in Figure 8.d.

They reveal that, for the smallest database, 22 attribute modifications and 30 insertions can be propagated incrementally for the cost of evaluating a query against the corresponding virtual view. With an increase in database size the ratios continue to grow.

In summary, IVM seems to be more beneficial with larger databases. For the examples, query answering over MVs is always better than over virtual views, no matter the database size, and the difference increases as size increases. The same behaviour is observed if one pitches incremental maintenance against rematerizalization. The benefits for both our small and large databases indicate that a significant number of update events can be carried out for the cost of answering the query over a non-materialized view.

5 Conclusions

The recent increase in research on identifying which views should be materialized indicates the importance of studies into the performance of incremental maintenance techniques. The most comprehensive study we can find of the performance of IVM is that of Hanson [8], which is an analytical model of immediate and deferred update propagation in relational databases. We are not aware of any research that has sought to validate the models presented by Hanson experimentally, or that has systematically assessed the performance of view maintenance techniques in relational (or object-oriented) systems. Although analytical models have been proposed in an object-oriented setting (e.g., [9]), we are aware neither of any thorough performance studies based on such models nor of comprehensive empirical studies of views in object databases. Again as an example, [9] provides a few illustrative examples of the performance of MultiView, but could not be said to have carried out a systematic study.

We believe, therefore, that this paper presents the most thorough experimental performance evaluation of materialized views to date. Experiments have been conducted that assess the performance of query answering and update propagation for different selectivities, query complexities and database sizes. It is clear from the experiments, as would be expected, that queries over materialized views generally run very much faster than those over virtual views. This observation, however, is not what has driven research in IVM. Research in IVM is predicated on two hypothesis:

1. That incremental propagation of updates can be expected to perform significantly better than rematerialization.
2. That incremental propagation of updates can provide improved performance overall for transactions that mix queries and updates, as long as updates are not dominant.

The experiments reported in this paper broadly support (1), with incremental update propagation generally being significantly faster than rematerialization. However, in terms of (2), although some results reported in the paper indicate that many incremental updates can be propagated in the time it takes to evaluate the corresponding virtual view, there are other cases in which the cost of propagating a single update is almost as great as the cost of evaluating the view. Thus this paper can be seen as enforcing the need for judicious selection of views for materialization, with many different issues impacting the decision making process.

Acknowledgements. M. Akhtar Ali gratefully acknowledges the support of the Commonwealth Scholarship Commission in the United Kingdom (Grant PK0257). We would like to thank Leonidas Fegaras for making available to the research community the λ-DB system and for helping us make use of it.

References

1. M. A. Ali, A. A. A. Fernandes, and N. W. Paton. Incremental Maintenance of Materialized OQL Views. In *Proc. 3rd DOLAP*, pages 41–48. ACM Press, 2000.
2. M. Carey, D.J. DeWitt, and J.F. Naughton. The OO7 Benchmark. In *Proc. SIGMOD*, pages 12–21, 1993.
3. R. G. G. Cattell, editor. *The Object Database Standard: ODMG 3.0.* Morgan Kaufmann, 2000.
4. G. K. Y. Chan, Q. Li, and L. Feng. Design and Selection of Materialized Views in a Data Warehousing Environment: A Case Study. In *Proc. DOLAP*, pages 42–47, 1999.
5. S. Cluet and C. Delobel. A General Framework for the Optimization of Object-Oriented Queries. In *Proc. SIGMOD*, pages 383–392, 1992.
6. L. Fegaras and D. Maier. Optimizing Object Queries Using an Effective Calculus. *ACM TODS*, 25(4), 2000.
7. A. Gupta and I. S. Mumick, editors. *Materialized Views: Techniques, Implementations, and Applications.* The MIT Press, 1999. ISBN 0-262-57122-6.
8. Eric N. Hanson. A Performance Analysis of View Materialization Strategies. In ACM Press, editor, *Proc. SIGMOD*, pages 440–453, 1987.
9. H. Kuno and E. Rudensteiner. Incremental Maintenance of Materialized Object-Oriented Views in MultiView: Strategies and Performance Evaluation. *IEEE TKDE*, 10(5):768–792, 1998.
10. C. Zhang and J. Yang. Genetic Algorithm for Materialized View Selection in Data Warehouse Environments. In *Proc. DaWaK*, pages 116–125, 1999.

Managing Time Consistency for Active Data Warehouse Environments

Robert M. Bruckner and A.M. Tjoa

Institute of Software Technology
Vienna University of Technology
Favoritenstr. 9-11 /188, A-1040 Vienna, Austria
{bruckner, tjoa}@ifs.tuwien.ac.at

Abstract. Real-world changes are generally discovered delayed by computer systems. The typical update patterns for traditional data warehouses on an overnight or even weekly basis enlarge this propagation delay until the information is available to knowledge workers. Typically, traditional data warehouses focus on summarized data (at some level) rather than detail data. For active data warehouse environments, also detailed data about individual entities are required for checking the data conditions and triggering actions. Hence, keeping data current and consistent in that context is not an easy task. In this paper we present an approach for modeling conceptual time consistency problems and introduce a data model that deals with timely delays. It supports knowledge workers, to find out, why (or why not) an active system responded to a certain state of the data. Therefore the model enables analytical processing of detail data (enhanced by valid time) based on a knowledge state at a specified instant of time. All states that were not yet knowable to the system at that point in time are consistently ignored.

1. Introduction

The observation of real-world events by computer systems is characterized by a delay. This so-called *propagation delay* is the time interval it takes for a monitoring system to realize an occurred state change. While operational systems are among other things designed to meet well-specified response time requirements, the focus of data warehouses (DWHs) [8] is generally the strategic analysis of data integrated from heterogeneous systems. The data integration process is very complex and covers the acquisition, extraction, transformation (staging area), and loading of the data into the DWH. Traditionally there is no real-time connection between a DWH and its data sources, because the write-once read-many decision support characteristics would conflict with the continuous update workload of operational systems and cause poor analysis concurrency. Data loading is done during frequent update windows (e.g. once a week, every night), when the analysis capabilities of DWHs are not affected.

Consequently, up till recently, *timeliness* requirements [7] (the relative availability of data to support a given process within the timetable required to perform the process) were restricted to mid-term or long-term. W. H. Inmon, known as the founder of data warehousing, cites time variance [8] as one of four silent characteristics of a DWH. Timeliness can be viewed as an aspect of data quality, which has a strong influence on the delay until a system realizes a certain state of the data. While a complete, real-time enterprise DWH might still be the panacea, there are some approaches to enable DWHs to react "just-in-time" [17] to changing customer needs, supply chain demands, and financial concerns.

New concepts, like *active data warehousing* [14], try to combine active (information) mechanisms based on ECA (event-condition-action) rules [1] with the integrated analysis power of DWH environments to extend (passive) systems with reactive capabilities: An active data warehouse (*ADWH*) is *event driven*, *reacts* in a timeframe appropriate to the business needs, and makes *tactical* decisions or causes operational actions rather than waiting for periodic reports. Therefore the design of an ADWH has to consider technical aspects (scalability, high availability, "just-in-time" data loading, mixed workload, etc.) as well as the integration of active mechanisms, which have to deal with three kinds of propagation delays in DWH environments:
1) Real-world changes are usually discovered and captured delayed by computer systems. 2) The typical update patterns for traditional DWHs on a weekly or even monthly basis enlarge these propagation delays. 3) Knowledge workers require consolidated and aggregated information for analysis purposes. However, in the case of *traditional DWHs* any significant delay in the recognition of real-world events may result in a number of additional considerations needing to be taken into account:

- *data integration.* "Old" aggregates have to be updated.
- *analytical processing.* Historical analysis results are not repeatable any longer, if additional information regarding that time period is integrated delayed (measures and summaries will change unexpectedly from the user's perspective).

The complexity of analysis and data access required places the typical ADWH query outside the scope of most transaction-processing systems and addresses a type of automated decision-making that has rarely been accomplished by passive DWHs [14].

Consequently, keeping data current and consistent in that context is not easy. Although higher refresh frequencies shrink the propagation delays, time consistency becomes even more important, because business rules can be formulated according to the ECA structure of triggers in ADWHs. Delays in the execution of such rules can cause unexpected and undesired side-effects [11].

We will concentrate on the development and examination of a schema model that copes with delays and allows a timely consistent representation of information. This distinction has a particular interest in DWHs, where it can be useful to consider information validity of detail data (or even aggregates), that is typically restricted to time periods, because of frequent updates. It is important to note that the aim of our approach is not necessarily to shrink delays. Rather, it is a pragmatic one to ensure that the side-effects of propagation delays are reduced in ADWHs. The main contribution of this paper is the introduction of overlapping valid time intervals.

The remainder of this paper is organized as follows. Section 2 considers research issues and related work. Time consistency is introduced in Section 3 and described from different viewpoints (conceptual, logical, and physical). Section 4 describes a case study followed by an assessment. Finally, we give a conclusion in Section 5.

2. Research Issue and Related Work

The importance of data quality for organizational success has been underestimated for a long time. Quality management in information processing is not nearly as established as it is in manufacturing [2]. Since DWHs store gigabytes up to terabytes of data, a manual quality control is not feasible at all. In data quality research, MIT's *Total Data Quality Management* (TDQM) [16] project considers *timeliness* as a so-called quality dimension of the *contextual* data quality.

An interesting practice approach is described in [7], where timeliness is viewed as the time from when a fact is *first captured* in an electronic format and when it is actually *knowable* by a knowledge worker who needs it. For data captured in a source database, uploaded to a central DWH over the weekend, the propagation delay can be several days. This is definitely unacceptable for implementations tightly integrated into business processes (like ADWHs), which are networked with other organizations.

Temporal databases provide support for past, current, or even future data and allow sophisticated queries over time to be stated [6, 19]. Research in temporal databases has grown immensely in recent years. In particular, transaction time and valid time have been proposed [9] and investigated [10, 15].

In the field of data warehousing [3] concentrated on the shortcomings of star schemas in the context of slowly changing dimensions. The conclusion is that state-oriented warehouses allow easier analytical processing and even better query performance than observed in regular events warehouses. Our formal approach to managing time consistency (described below) is state-oriented, too. Temporal data warehouses (in particular temporal view self-maintenance) are addressed by [18].

Recent research in *active databases* [5] aims at extending the power of active rules (for responding to changes to the database and to external events) to trigger on complex patterns of temporal events. An important issue is the accommodation of *delays*, which is investigated for *temporal active databases* in [11]. This paper concentrates on delays in the quite young discipline of *active data warehouses* [14].

3. Time Consistency

The notion of time is fundamental to our existence and an important aspect of real-world phenomena. We can reflect on past events and on possible future events, and thus reason about events in the domain of time. In many models, time is an independent variable that determines the sequence of *states* of a system. The continuum of real time can be modeled by a *directed timeline* consisting of an infinite set {T} of *instants* (time points on an underlying time axis [9]). A section of the timeline is called a *duration* or *time period.* An *event* takes place at an instant of time, and therefore does not have duration. If two events occur at an identical instant, then the two events are said to occur *simultaneously.* A *time interval* is determined by the duration between two corresponding (start - end) instants.

Figure 1 below describes a situation, where a computer system observes the state S_1 at T_1 that a specified person (Mr. Smith) lives in Boston. It knows (or assumes – that does not matter for the model) that Mr. Smith stays there from T_{1S} to T_{1E}. The next state (S_2) observed by the computer system regarding Mr. Smith is the new address in New York at T_2. Furthermore, additionally captured information reveals that Mr.

Smith already lived in New York since T_{2S}. By modeling this situation timely consistent, it will always be possible to find out, that the system did *not* know where Mr. Smith lived after T_{1E} until the instant T_2, when the new piece of information (state S_2) was integrated. Based on this knowledge an ADWH can determine the "right" strategy e.g. for *customer relationship management* (CRM): An "information package for new citizens of NY" is appropriate at T_{2S}, but may be unsuitable at the instant T_2, when the real-world event was actually knowable to the ADWH.

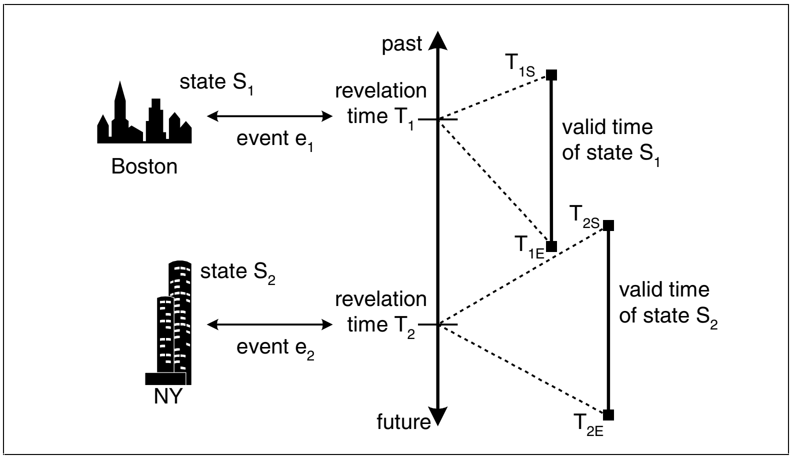


Fig. 1. Sequencing the changes to capture the history regarding an entity of interest.

3.1. A Conceptual Model for Time Consistency

A timely consistent representation of information requires a reliable view on historical data at any point in time independent from propagation delays.

A *knowledge state* (*KS*) is determined by a specified instant T. It considers all information (knowledge) that was observed, captured, and integrated until T. An ordered relation of two instants $T_1 < T_2$ implies that $KS(T_1)$ $KS(T_2)$. In other words, moving forward in time causes the knowledge state to grow. In general an analysis focuses on a time interval containing at least one *instant of interest* (*II*): $II_{interest} = [II_{start}, II_{end}]$ (e.g. May 1st, 2001). The KS and II are two *orthogonal* time dimensions and therefore independent from each other regarding analysis capabilities.

A data model enables *time consistency* if a set of nine conditions listed in Table 1 is satisfied for *any* combination of two stored datasets (S_1, S_2) regarding the *same subject*. In general a stored state S_X is determined by an instant T_X (*revelation time*) and a corresponding *valid time* indicated by a time interval $[T_{XS}, T_{XE}]$.

Any combination of II with KS during analytical processing has to retrieve a correct state, which is actually knowable to the system at the specified KS. The *retrieved state* regarding an entity of interest will be S_1, or S_2, or "*not defined*" (if it is neither S_1 nor S_2). Table 1 (which is related to Figure 1) describes the timely correct state that should be retrieved during analytical processing. The retrieved state can be viewed as the output of a function considering the applied KS and the specified II.

The conceptual model is able to handle past and future validity periods (e.g. instant(T_1) > instant(T_{1E}), instant(T_1) < instant(T_{1S})), which is important 1) to deal with propagation delays regarding timeliness, and 2) to support planning. The reading examples below Table 1 exactly describe the major contributions of this conceptual model to conventional temporal models restricted to non-overlapping valid times.

Table 1. Conceptual model for time consistency.

Knowledge State (*KS*)	**Instant(s) of interest (*II*)**	**Retrieved state**[1]
KS < T_1	Any **II**	(not defined)[2]
T_1 **KS** < T_2	**II** < T_{1S}	(not defined)
	T_{1S} **II** T_{1E}	**S_1**
	II > T_{1E}	(not defined)
KS T_2	**II** < T_{1S}	(not defined)
	T_{1S} **II** < min(T_{1E} , T_{2S})	**S_1**
	T_{1E} < **II** < T_{2S}[3]	(not defined)
	min(T_{1E} , T_{2S}) **II** T_{2E}	**S_2**
	II > T_{2E}	(not defined)

Reading examples. At any point in time an analysis based on a KS between T_1 and T_2 (T_1 KS < T_2) is able to figure out 1) the state S_1 was valid till T_{1E} (third entry in Table 1 above), and 2) S_2 was not yet knowable to the system (fourth entry).

Moving forward in time (by applying a knowledge state of KS T_2) will reveal that e.g. the state S_1 was only valid till T_{2S} (sixth entry in Table 1), under the assumption of overlapping valid times for S_1 and S_2.

3.2. Modeling Temporal Consistency for Data Warehouses

The issue of modeling time consistency for centralized DWH environments is challenging due to the following reasons:

1. DWHs have to deal with an additional propagation delay regarding electronically captured data by their source systems (described in Section 1).
2. The integration of distributed source systems (e.g. different time zones, delays in data processing) is necessary during data staging.

In our opinion, the most efficient approach is to separate these two issues as described above by extending the logical and physical data model to handle the former one. The latter step, which is complicated due to temporal order issues (see Section 3.3), is done during data staging and therefore does not affect analytical processing.

We propose an enhanced time dimension model for DWHs to accomplish time consistency, because simply adding timestamps during data loading on an overnight or even weekly basis is not precise enough to model the instant, when a new fact was first captured electronically and therefore knowable to an organization. The identified propagation delays in a ADWH implementation are represented as follows:

[1] The retrieved state is timely correct regarding the electronically captured input data.

[2] "not defined" means neither S_1 nor S_2.

[3] This case is obsolete, if the corresponding valid times [T_{1S}, T_{1E}] and [T_{2S}, T_{2E}] do not overlap.

- **valid time** (validity of knowledge). This property is always related to a base event at an instant T (stored as dataset), and describes the time interval of validity (knowable at instant T) of that piece of information.
- **revelation time** (*transaction time*). The revelation time describes the point in time, when a piece of information was realized by at least one source system. This concept is tightly related to the notion of transaction time [9, 15] in temporal databases. However, in the context of ADWHs we call this property "revelation time" to clarify that 1) transaction results were already recorded in the sources and 2) analytical processing is proposed to reveal interesting relations among facts.
- **load timestamp**. The load timestamp is a time value associated with some integrated dataset in a DWH environment and determines the instant, when a fact is actually knowable to *active* mechanisms. Thus, in the presence of delays, it is the load timestamp that represents the point in time to which automatic decision making or compensating actions should refer in ADWH environments.

The conceptual model is associated with the mentioned (orthogonal) types of time information as follows: The *valid time* is related to the *instants of interest (II)* during analytical processing. Typically the *knowledge state (KS)* relates to the *revelation time*, to get a timely accurate picture of the entity of interest. However, it is possible to relate to the *load timestamp*, to enable knowledge workers to observe and control active mechanisms in an ADWH.

Logical model:
When designing a temporal data model, an important and central aspect is the choice of appropriate timestamps of the database facts. Time intervals [4] are used as an abbreviation for sets of points for practical reasons. A single timestamp approach storing only a start time when a record became valid is well applicable to event data, but faces serious deficiencies in the context of DWHs, where in general state information is stored. Simply returning such temporary values to users during analytical processing *without* integration into the data model may lead to confusion.

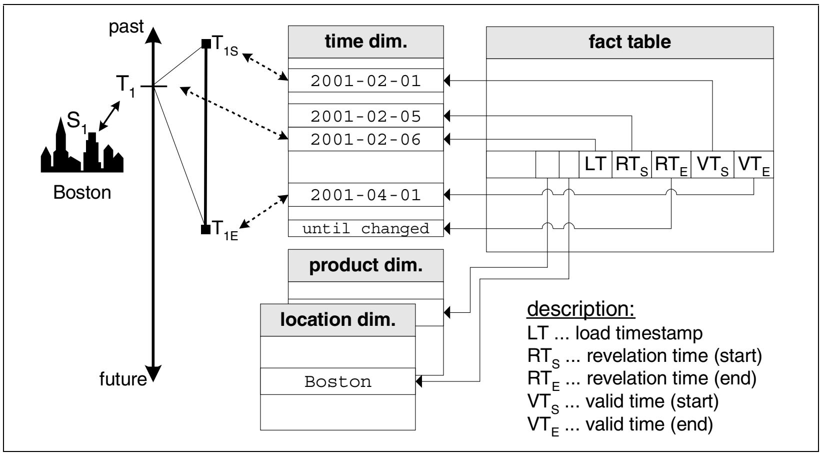


Fig. 2. Representation of state S_1. This piece of information was revealed on February 5th with a valid time from February 1st to April 1st and loaded into the ADWH on February 6th.

In order to model past and future instants two symbolic instants are introduced: – ("since ever") and + ("until changed"). We will reuse the time dimension commonly available in all warehouse models to integrate the *load timestamps* of facts (as described above). Thus analytical processing known from traditional DWHs is effectively supported. Both, the *revelation* and *valid time* of facts will be integrated as separated logical time dimensions modeled by time intervals. The physical implementation is done by a bitemporal model (exemplified in Figure 2) combining the features of a transaction-time and a valid-time database. Therefore it represents reality more accurately and allows for retroactive as well as postactive changes.

The proposed approach is suitable to on-line analytical processing (OLAP), supporting the creation of cubes well: e.g. analyzing the average propagation delays (differences between valid and revelation times) by drilling down to the distributed operational systems located in different geographical regions.

Time consistency for *aggregates* is handled the same way as for facts. The integration of new facts concerning information about previous data loads, causes the re-computation of the affected aggregates. The "old" aggregates can either be deleted or stored time consistently by adjusting the valid time from "until changed" to the re-computation timestamp.

Physical model:
The temporal logical model presented above permits the designer to mix temporal and nontemporal aspects. In particular it is possible to apply different strategies to every *fact star* in a DWH. Actually implementing a logical data model supporting time consistency is influenced by following issues:

- It is necessary to provide a *physical representation* for symbolic instants (+ , –), e.g. by pre-defined date constants.
- Indexes are even more important in temporal relations due to their size. Furthermore, the focus of temporal analytical processing on *coalescing* (merging value-equivalent tuples with intervals that overlap or meet) and *temporal joins* [10] requires appropriate indexes [13].
- In temporal queries, conjunctions of inequality predicates (which are harder to optimize) appear more frequently. Consequently, improved cost models for the optimization of temporal operators have to be applied.

3.3. Data Staging

While some source systems may store partial history, many nontemporal sources store only the current state of their data. Even if DWH storage is not a concern, it is helpful to identify those sources/data where delays possibly occur that have side-effects on automatic decision making in ADWHs and thus should be modeled in a time consistent manner. Whereas data staging in traditional DWHs establishes a *delivery order* for new information, managing time consistency requires the (stronger) *temporal order*, which takes into account already stored facts and aggregates.

During data staging we have to consider three issues in order to get a single version of truth regarding time during data staging: 1) different time zones of distributed sources, 2) differences in observing the same event by different operational systems; e.g. a CRM system records an order cancellation on May 2nd, whereas the billing system observes it on May 3rd, and 3) propagation delays, as described in Section 1.

The former two issues are addressed by establishing a global time base known from the field of distributed or real-time systems. The latter one is managed by the proposed approach to time consistency modeling: The integration of new data enhancing the knowledge state regarding a particular subject is done by *retroactive updates*. Existing data structures (in particular the end date for revelation time) of the ADWH only have to be modified if valid times overlap. It is important to note that both the valid times of the old state provided by ADWH and the new one from data staging (provided by temporal source systems) are *not* changed.

Table 2 shows the physical datasets according to the situation exemplified in Figure 1 and Figure 2. This kind of temporal integration method slows down data staging in the case of overlapping valid times, but simplifies analytical processing dramatically. As mentioned earlier this method is applicable to aggregates, too.

Table 2. Temporal order for overlapping valid times.

location	valid time	revelation time	load timestamp
Boston	2001-02-01 – 2001-04-01	2001-02-05 – until changed	2001-02-06

location	valid time	revelation time	load timestamp
Boston	2001-02-01 – 2001-04-01	2001-02-05 – **2001-04-14**	2001-02-06
NY	2001-03-15 – 2001-06-01	**2001-04-15** – until changed	2001-04-16

4. Experimental Results

The usability of the proposed approach was evaluated by the application to an industry project. We found out that the introduction of the knowledge state viewpoint is actually an enhancement to analytical processing. Furthermore, *planning* and *monitoring* including *what-if scenarios* (i.e. different versions of plans) are supported at various aggregation levels, because valid times are suitable to past, present and future states. The focus of our approach to managing time consistency enables knowledge workers to establish and control active behavior for ADWH environments.

Case study: PLUS project

The PLUS project was an industry project in the domain of accident insurance implemented at a big Austrian insurance authority starting in 1998 and continued till 2000. One focus of the project was the management of time consistency and the introduction of *active behavior* to automate routine decision tasks. For example, initiate and control the stopping of accident pension payments due to pre-specified reasons (successful rehabilitation, death, etc.). From a technical viewpoint, the project implemented a *data mart* (~ 5 GB) on an Oracle database tightly integrated with a workflow management system and the organization's billing system. The data model contained fact stars using the proposed temporal model, as well as non-temporal fact stars. The source systems were partly enhanced to provide additional temporal information to better deal with delays. Legacy data was integrated using valid times reconstructed from the sequence of states available regarding the same subject.

A very important analytical application for public insurance authorities is the periodic generation of many *timely consistent* statistical reports classifying and investigating changes of accident pensions from various viewpoints (due to official statistics for the European Statistical System). The approach to manage time consistency reduced the necessary effort to generate and maintain statistics dramatically. Generating such statistics without temporal support is difficult, because every report has to consistently contain aggregated figures concerning previous months or years.

Another driving force to establish this kind of temporal data marts is the limitation of traditional DWHs regarding e.g. cash flows, which differ from reality, if significant changes are captured delayed. For example, a person has an accident on June 29th and thus will receive a pension. Due to typical delays (pension request, data processing, weekends, etc.) the decision is made on July 5th, when both the billing and statistics for June is already completed. Traditional DWHs ignoring the revelation time and thus storing June 29th, will invalidate cash flows and statistics (as already mentioned).

Assessment

Managing time consistency according to the presented *state-oriented* approach allows us to achieve consistent analysis results at any point in time for a specified knowledge state. In particular, the approach

+ is based on a formal model with strong (but simple) guidelines. The implementation uses well-understood concepts, e.g. bitemporal tables [3, 15].
+ enhances the time dimension and thus is suitable to various application domains.
+ improves data quality during data staging by considering stored data structures.
+ can be introduced at any point in time. If temporal information is not available for legacy data, regular valid and revelation time intervals ("until changed") will be used to "upgrade" the old data structures.
+ enhances analytical capabilities of DWHs (knowledge state, instants of interest).
+ does not restrict the entire data warehouse to be temporal, but rather permits the designer to mix temporal and nontemporal aspects.

covers only time relevant aspects of data quality management in DWHs.

needs to consider performance issues: Updates to time attributes of existing fact-table rows are necessary when bulk loading the ADWH.

In contrast, an *event-oriented* DWH (each row in the fact table represents the *change* occurred according to the previous event) is easier to feed with new data. However, analytical processing is slowed down, because queries asking questions about states of some object in some period of time will result in the comparison of timestamps from different rows.

5. Conclusion and Further Research

The hypercube-based multidimensional model, and the star-schema based (extended-) relational model have emerged as candidate data models for DWHs. However, these models do not adequately address issues related to history data and time consistency, which are certainly core issues in data warehousing, particularly for ADWHs.

The advantages provided by built-in time consistency support in data warehouses include higher-fidelity data modeling, more efficient capturing of an organizations history, as well as analyzing the sequence of changes to that history.

Overall, we have shown how to provide knowledge workers with simple, consistent, and efficient support for ADWH environments. The presented approach enables a time consistent view for analysis purposes considering that the validity of detail data (or aggregates) is typically restricted to time periods, because of frequent updates and capturing delays. Ignoring this temporal issues leads to impoverished expressive power and questionable query semantics in many real-life scenarios.

Important future research directions in this field will be the maintenance of DWHs over dynamic information systems (data updates, schema changes, dynamic sources) [12], and enhancements to the active behavior (e.g. *automatic* compensation actions for significant changes) in the field of ADWH.

References

1. The ACT-NET Consortium: *The Active Database Management System Manifesto: A Rulebase of ADBMS Features*. In ACM SIGMOD Record, Vol.25(3): pp. 414-471, 1996.
2. Ballou, D.P., Tayi, G.K.: *Enhancing Data Quality in Data Warehouse Environments.* Communications of the ACM, Vol. 42(1): pp. 73-78, 1999.
3. Bliujute, R., Saltenis, S., Slivinskas, G., Jensen, C.S.: *Systematic Change Management in Dimensional Data Warehousing*. Technical Report TR-23, TIMECENTER, January 1998.
4. Böhlen, M.H., Busatto, R., Jensen, C.S.: *Point- Versus Interval-Based Temporal Data Models*. Proc. of 14th Intl. Conf. ICDE, IEEE Comp. Society Press, pp. 192-201, Orlando, USA, 1998.
5. Dittrich, K.R., Gatziu, St.: *Aktive Datenbanksysteme*. Dpunkt. Verlag, Heidelberg, 2000.
6. Dyreson, C.E., Evans, W.S., Hong, L., Snodgrass, R.T.: *Efficiently Supporting Time Granularities*. In IEEE Trans. on Knowledge and Data Engineer., Vol. 12(4): pp. 568-587, 2000.
7. English, L.P.: *Improving Data Warehouse and Business Information Quality.* John Wiley and Sons, New York, 1999.
8. Inmon, W.H.: *Building the Data Warehouse*. John Wiley and Sons, New York, 1996.
9. Jensen, C.S., Dyreson, C.E., (eds): *The Consensus Glossary of Temporal Database Concepts*. In *Temporal Databases: Research and Practice*, Springer, LNCS 1399, pp. 367-405, 1998.
10. Jensen, C.S., Snodgrass, R.T.: *Temporal Data Management*. In IEEE Transactions on Knowledge and Data Engineering, Vol. 11(1): pp. 36-44, 1999.
11. Roddick, J.F., Schrefl, M.: *Towards an Accommodation of Delay in Temporal Active Databases*. Proceedings of the 11th Australasien Database Conference (ADC2000), IEEE Computer Society, pp. 115-119, Canberra, Australia, 2000.
12. Rundensteiner, E.A., Koeller, A., Zhang, X.: *Maintaining Data Warehouses over Changing Information Sources*. Communications of the ACM, Vol. 43(6): pp. 57-62, 2000.
13. Salzberg, B., Tsotras, V.J.: *Comparison of access methods for time-evolving data*. In ACM Computing Surveys, Vol. 31(2): pp. 158-221, 1999.
14. Schrefl, M., Thalhammer, T.: *On Making Data Warehouses Active*. Proceedings of the 2nd International Conference DaWaK, Springer, LNCS 1874, pp. 34-46, London, UK, 2000.
15. Torp, K., Jensen, C.S., Snodgrass, R.T.: *Effective Timestamping in Databases*. The VLDB Journal, Vol. 8, Issue 3-4, pp. 267-288, 2000.
16. Wang, R.Y.: *Total Data Quality Management*. Communications of the ACM, Vol. 41(2): pp. 58-65, 1998.
17. Westerman, P.: *Data Warehousing: Using the Wal-Mart Model*. Morgan Kaufmann Publishers, San Francisco, 2001.
18. Yang, J., Widom, J.: *Temporal View Self-Maintenance*. Proceedings of the 8th Intl. Conf. EDBT2000, Springer, LNCS 1777, pp. 395-412, Konstanz, Germany, 2000.
19. Zaniolo, C., Ceri, S., Faloutsos, C., Snodgrass, R.T., Subrahmanian, V.S., Zicari, R.: *Advanced Database Systems*. Morgan Kaufmann Publishers, San Francisco, 1997.

Optimization Algorithms for Simultaneous Multidimensional Queries in OLAP Environments

Panos Kalnis and Dimitris Papadias

Department of Computer Science
Hong Kong University of Science and Technology
Clear Water Bay, Hong Kong
`http://www.cs.ust.hk/{~kalnis, ~dimitris}`

Abstract. Multi-Dimensional Expressions (MDX) provide an interface for asking several related OLAP queries simultaneously. An interesting problem is how to optimize the execution of an MDX query, given that most data warehouses maintain a set of redundant materialized views to accelerate OLAP operations. A number of greedy and approximation algorithms have been proposed for different versions of the problem. In this paper we evaluate experimentally their performance using the APB and TPC-H benchmarks, concluding that they do not scale well for realistic workloads. Motivated by this fact, we developed two novel greedy algorithms. Our algorithms construct the execution plan in a top-down manner by identifying in each step the most beneficial view, instead of finding the most promising query. We show by extensive experimentation that our methods outperform the existing ones in most cases.

1 Introduction

Data warehouses have been successfully employed for assisting decision-making by offering a global view of the enterprise data and providing mechanisms for On-Line Analytical Processing (OLAP) [CCS93]. A common technique to accelerate OLAP operations is to store some redundant data, either statically [Gupt97, GM99, SDN98] or dynamically [KR99].

Most of OLAP literature assumes that queries are sent to the system one at a time. In multi-user environments, however, many queries can be submitted concurrently. In addition, the new API proposed by Microsoft [MS] for Multi-Dimensional Expressions (MDX), which becomes de-facto standard for many products, allows the user to formulate multiple OLAP operations in a single MDX expression. For a set of OLAP queries, an optimized execution plan can be constructed to minimize the total execution time, given a set of materialized views. This is similar to the multiple query optimization problem for general SQL queries [PS88, S88, SS94, RSSB00], but due to the restricted nature of the problem, better techniques can be developed.

Zhao et al. [ZDNS98] were the first ones to deal with the problem of multiple query optimization in OLAP environments. They designed three new join operators, namely: *Shared scan for Hash-based Star Join*, *Shared Index Join* and *Shared Scan for Hash-based and Index-based Star Join.* These operators are based on common subtask sharing among the simultaneous OLAP queries. Such subtasks include the scanning of the base tables, the creation of hash tables for hash based joins, or the filtering of the base tables in the case of index based joins. Their results indicate that there are substantial savings by using these operators in relational database systems. In the same paper they propose three greedy algorithms for creating the optimized execution plan for an MDX query, using the new join operators. Liang et. al [LOY00] also present approximation algorithms for the same problem.

In this paper we use the TPC-H [TPC] and APB [APB] benchmarks in addition to a 10-dimensional synthetic database, to test the performance of the above-mentioned algorithms under a realistic workload. Our experimental results suggest that the existing algorithms do not scale well when we relax the constraints for the view selection problem. We observed that in many cases when we allowed more space for materialized views, the execution cost of the plan derived by the optimization algorithms was higher than the case where no materialization was allowed.

Motivated by this fact, we propose a novel greedy algorithm, named *Best View First* (*BVF*) that doesn't suffer from this problem. Our algorithm follows a top-down approach by trying to identify the most beneficial view in each iteration, as opposed to finding the most promising query to add to the execution plan. Although the performance of *BVF* is very good in the general case, it deteriorates when the number of materialized views is small. To avoid this, we also propose a multilevel version of *BVF* (*MBVF*). We show by extensive experimentation that our methods outperform the existing ones in most realistic cases.

The rest of the paper is organized as follows: In section 2 we introduce some basic concepts and we review the work of [ZDNS98] and [LOY00]. In section 3 we identify the drawbacks of the current approaches and in section 4 we describe our methods. Section 5 presents our experimental results while section 6 summarizes our conclusions.

2 Background

A multidimensional expression (MDX) [MS] provides a common interface for decision support applications to communicate with OLAP servers. Here we are interested on the feature of expressing several related OLAP queries with a single MDX expression. Therefore, an MDX expression can be decomposed into a set Q of group-by SQL queries. The intuition behind optimizing an MDX expression is to construct subsets of Q that share star joins, assuming a star schema for the warehouse. Usually, when the selectivity of the queries is low, hash-based star joins [Su96] are used; otherwise, the index-based star join method [OQ97] can be applied. [ZDNS98] introduced three shared join operators to perform the star joins.

The first operator is the *shared scan for hash-based star join*. Let q_1 and q_2 be two queries which can be answered by the same materialized view v. Consequently they will share some (or all) of their dimensions. Assume that both queries are non-selective, so hash-based join is used. To answer q_1 we construct hash tables for its dimensions and we probe each tuple of v against the hash tables. Observe that for q_2 we don't need to rebuild the hash tables for the common dimensions. Furthermore, only one scanning of v is necessary. Consider now that we have a set Q of queries all of which use hash-based star join and let L be the lattice of the data-cube and MV be the set of materialized views. We want to assign each $q \in Q$ to a view $v \in MV$ such that the total execution time is minimized. If v is used by at least one query, its contribution to the total execution cost is:

$$t^{hash}_{MV}(v) = Size(v)\ \ t_{I/O} + t_{hash\_join}(v) \quad \textbf{(1)}$$

where *Size(v)* is the number of tuples in v, $t_{I/O}$ is the time to fetch a tuple from the disk to the main memory, and $t_{hash\_join}(v)$ is the total time to generate the hash tables for the dimensions of v and to perform the hash join. Let q be a query that is answered by $v \equiv mv(q)$. Then the total execution cost is increased by:

$$t^{hash}_{Q}(q,mv(q)) = Size(mv(q))\ \ t_{CPU}(q,mv(q)) \quad \textbf{(2)}$$

where $t_{CPU}(q,v)$ is the time per tuple to process the selections in q and to evaluate the aggregate function. Let $MV' \subseteq MV$ be the set of materialized views which are selected to answer the queries in Q. The total cost of the execution plan is:

$$t^{hash}_{total} = \underset{v\ \ MV'}{\ \ t^{hash}_{MV}(v)} + \underset{q\ \ Q, mv(q)\ \ MV'}{\ \ t^{hash}_{Q}(q,mv(q))} \quad \textbf{(3)}$$

The problem of finding the optimal execution plan is equivalent to minimizing t^{hash}_{total} which is likely to be NP-hard. [LOY00] provide an exhaustive algorithm which runs in exponential time. Since the algorithm is impractical for real life applications, they also describe an approximation algorithm. They reduce the problem to a directed Steiner tree problem and apply the algorithm of Zelikovsky [Zeli97]. The solution is $O(|Q|^{\varepsilon})$ times worse than the optimal, where $0 < \varepsilon \leq 1$.

The second operator is the *shared scan index-based join*. Similar to the previous case, a set of queries are answered by the same view v, but there are bitmap indexes that can be used to accelerate all queries. The read and execution cost are defined as before, the only difference being that they are scaled according to the selectivity of the indexes (see [KP00] for details). The aim is to minimize the total cost t^{index}_{total}. In addition to an exact exponential method, [LOY00] propose a polynomial approximation algorithm that delivers a plan whose execution cost is $O(|Q|)$ times the optimal.

The third operator is the *shared scan for hash-based and index-based star joins*. As the name implies, this is a combination of the previous two cases. [ZDNS98] propose three heuristic algorithms to construct an execution plan: i) the *Two Phase Local Optimal* algorithm (*TPLO*) which constructs the optimal plan for each query and then merges the individual plans, ii) the *Extended Two Phase Local Greedy* algorithm (*ETPLG*) which constructs the execution plan incrementally, adding one query at a time and starting from the most general queries, and iii) the *Global Greedy* algorithm (*GG*) which is similar to *ETPLG* but allows modifications to the already constructed part of the plan. [LOY00] propose another algorithm, named *GG-c*, which is similar to *ETPLG* but the order that the queries are inserted is defined dynamically.

None of the above algorithms scales well, when the number of materialized views increases. In the next section we present an example that highlights the scalability problem and verify experimentally that it affects severely the performance of the algorithms under realistic workloads.

3 Motivation

Figure 1 shows an instance of the multiple query optimization problem where $\{v_1, v_2, v_3\}$ is the set of materialized views and $\{q_1, q_2\}$ are the queries. We assume for simplicity that all queries use hash-based star join. Let $t_{I/O} = 1$, $t_{hash\_join}(v) = Size(v)/10$ and $t_{CPU}(q,v) = 10^{-2}$, $\forall$ *q*, *v*. *GG-c* will search for the combination of queries and views that result in minimum increase of the total cost, so it will assign q_1 to v_1. At the next step q_2 will be assigned to v_2 resulting to a total cost of 277.5. Assume that v_3 is the fact table of the warehouse and v_1, v_2 are materialized views. If no materialization were allowed, *GG-c* would choose v_3 for both queries resulting to a cost of 224. We observe that by materializing redundant views, the performance deteriorates instead of improving. Similar examples can be also constructed for the other algorithms.

In order to evaluate this situation under realistic conditions, we employed datasets from the TPC-H benchmark [TPC], the APB benchmark [APB] and a 10-dimensional synthetic database (SYNTH), where the size of the fact table was 6M, 1.3M and 20M tuples, respectively (see [KP00] for details). Since there is no standard benchmark for MDX, we constructed a set of 100 synthetic MDX queries. Each of them can be analyzed into 2 sets of 2 related SQL group-by queries (*q2_2* query set). We used this relatively small query set, in order to be able to run an exhaustive algorithm and compare the cost of the plans with the optimal one. We varied the available space for the materialized views (S_{max}) from 0.01% to 10% of the size of the entire data cube (i.e. the case where all nodes in the lattice are materialized). We used the *GreedySelect* [HRU96] algorithm to select the set of materialized views.

We employed the shared operators and we compared the plans delivered by the optimization algorithms, against the optimal plan. All the queries used hash-based star join. We implemented the greedy algorithm of [ZDNS98] (*GG*) and the one of [LOY00] (*GG-c*). We also implemented the Steiner-tree-based approximation algorithm of [LOY00] for hash-based queries (*Steiner-1*). We set $\varepsilon = 1$, since for smaller values of ε the complexity of the algorithm increases while its performance doesn't change considerably, as the experiments of Liang et. al. suggest. For obtaining the optimal plan, we used an exhaustive algorithm whose running time (for $S_{max} = 10\%$) was 5300, 290 and 91 sec, for the SYNTH, the TPC-H and APB datasets respectively.

Although the query set is too small to make safe conclusions, we can identify the instability problems. There is a point where the cost of the execution plan increases although more materialized views are available. Moreover, we observed that for the SYNTH dataset, when S_{max} varied from 1% to 5%, the execution cost of the plans delivered by *GG*, *GG-c* and *Steiner-1*, is higher in the presence of materialized views

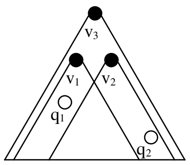
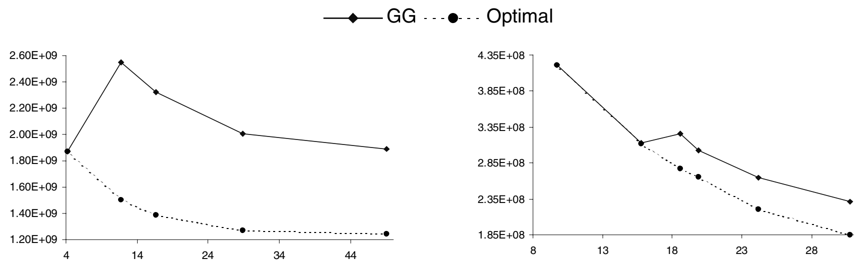


Fig. 1. Two instances of the multiple-query optimization problem.v_1|=100, $|v_2|$=150, $|v_3|$=200



(a) SYNTH dataset **(b)** TPC-H dataset

Fig. 2. Total execution cost versus *AVQ*

(i.e. we could achieve lower cost if we had executed the queries against the base tables).

The performance of the algorithms is affected by the *tightness* of the problem. Let *AVQ* be the average number of materialized views that can by used to answer each group-by query. We identify three regions: (i) The *high-tightness region* where the value of *AVQ* is small (i.e. very few views can answer each query). Since the search space is small, the algorithms can easily find a near optimal solution. (ii) The *low-tightness region* where *AVQ* is large. Here, each query can be answered by many views, so there are numerous possible execution plans. Therefore there exist many near-optimal plans and there is a high probability for the algorithms to choose one of them. (iii) The *hard-region*, which is between the high-tightness and the low-tightness regions. The problems in the hard region have a quite large number of solutions, but only few of them are close to the optimal, so it is difficult to locate one of these plans.

In figure 2 we draw the cost of the plan for *GG* and the optimal plan versus *AVQ*. For the SYNTH dataset the transition between the three regions is obvious. For TPC-H, observe that for small values of *AVQ*, the solution of *GG* is identical to the optimal one. Therefore the high execution cost for the plan is due to the specific instance of the problem. We can identify the hard region at the right part of the diagrams, when the trend for *GG* moves to the opposite direction of the optimal plan. Similar results were also observed for APB and for other query setsm, for all algortithms.

In summary, existing algorithms suffer from scalability problems, when the number of materialized views is increased. In the next section we will present two novel greedy algorithms, which have better behavior and outperform the existing ones in most cases.

4 Improved Algorithms

The intuition behind our first algorithm, named *Best View First* (BVF), is simple: Instead of constructing the global execution plan by adding the queries one by one (bottom-up approach), we use a top-down approach. At each iteration the most beneficial view *best_view* $\in$ *MV* is selected, based on a *savings* metric, and all the queries which are covered by *best_view* and have not been assigned to another view yet, are inserted in the global plan. The process continues until all queries are covered. Figure 3 shows the pseudocode of *BVF*.

```
ALGORITHM BVF(MV, Q)
/* MV:={v1,v2, …,v|MV|} is the set of materialized views */
/* Q:={q1,q2, …,q|Q|} is the set of queries */
AMV:=MV /* set of unassigned materialized views */
AQ:=Q /* set of unassigned queries */
GlobalPlan:=∅
while AQ≠∅
   Let best_view be the view with the highest savings value
   /* newSet is a set of queries that share an operator */
   newSet.answered_by_view:=best_view
   newSet.queries:= {q∈AQ: q is answered by best_view}
   GlobalPlan:=GlobalPlan ∪ newSet
   AMV:=AMV-best_view
   AQ:=AQ-{q∈AQ: q is answered by best_view}
endwhile
return GlobalPlan
```

Fig. 3. Best View First (BVF) greedy algorithm

The *savings* metric is defined as follows: Let $v \in MV$, and let $VQ \subseteq Q$ be the set of queries that can be answered by v. Let $C(q,u_i)$ be the cost of answering $q \in VQ$, by using $u_i \in MV$ and $C_{\min}(q) = \underset{1\ \ i\ \ |MV|}{\min}(C(q,u_i))$ that of answering q by using the most beneficial materialized view. Then

$$s\_\cos t(v) = \underset{q_i\ \ VQ}{\ } C_{\min}(q_i) \qquad \textbf{(4)}$$

is the best cost of answering all queries in VQ individually (i.e. without using any shared operator). Let *cost(v)* be the cost of executing all queries in VQ against v, by utilizing the shared operators. *savings(v)* equals to the difference between *s_cost(v)* and *cost(v)*.

The complexity of the algorithm is polynomial. To prove this, observe first that $C_{min}(q)$ can be calculated in constant time if we store the relevant information in the lattice during the process of materializing the set MV. Then *s_cost(v)* and *cost(v)* are calculated in $O(|VQ|) = O(|Q|)$ time in the worst case. The inner part of the for-loop is executed $O(|AMV|) = O(|MV|)$ times. The while-loop is executed $O(|Q|)$ times because in the worst case, only one query is extracted from AQ in each iteration. Therefore, the complexity of *BVF* is $O(|Q|^2 \cdot |MV|)$.

Theorem 1: *BVF* delivers an execution plan whose cost decreases monotonically when the number of materialized views increases. The proof can be found in the full version of the paper [KP00]. It follows that:

Lemma 1: *BVF* delivers an execution plan *P* whose cost is less or equal to the cost of executing all queries against the base tables of the warehouse by using shared star join.

Theorem 1 together with lemma 1, guarantee that *BVF* avoids the pitfalls of the previous algorithms. Note that there is no assurance for the performance of *BVF* compared to the optimal one, since the cost of answering all the queries from the base tables can be arbitrary far from the cost of the optimal plan. Consider again the example of figure 1, except that there are 100 queries that are answered by $\{v_1, v_3\}$ and 100 queries that are answered by $\{v_2, v_3\}$. *savings* for v_1 and v_2 is zero, while *savings*(v_3) = 11100 + 16650 – 620 = 27130, so all queries are assigned to v_3. The cost for the plan is 620. However, if we assign to v_1 all the queries that are bellow it and do the same for v_2, the cost of the plan is 525. We can make this example arbitrarily bad, by adding more queries bellow v_1 and v_2.

In general, *BVF* tends to construct a small number of sets, where each set contains many queries that share the same star join. This behavior usually results to high cost plans when there are a lot of queries and a small number of materialized views. To overcome this problem, we developed a multilevel version of *BVF*, called *MBVF*. The idea is that we can recursively explore the plan delivered by *BVF* by assigning some of the queries to views that are lower in the lattice (i.e. less general views) in order to lower the cost. *MBVF* works as follows (see [KP00]): First it calls *BVF* to produce an initial plan, called *LowerPlan*. Then, it selects from *LowerPlan* the view *v* which is higher in the lattice (i.e. the more general view). It assigns to *v* the queries that cannot be answered by any other view and calls *BVF* again for the remaining views and queries to produce *newPlan*. *v* and its assigned queries plus the *newPlan* compose the complete plan. If its cost is lower that the original plan, the process continues for *newPlan*, else the algorithm terminates. In the worst case, the algorithm will terminate after examining all the views. Therefore, the complexity is $O(|Q|^2 \cdot |MV|^2)$.

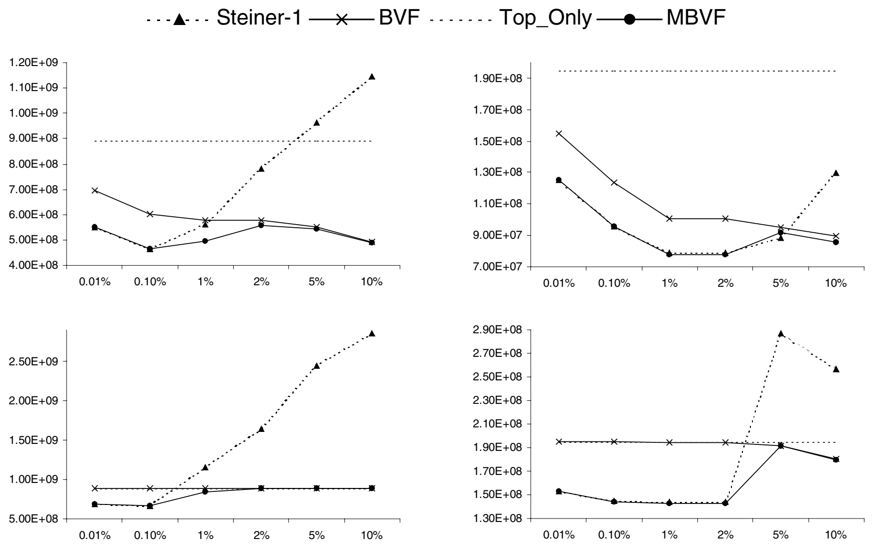
Lemma 2: The cost of the execution plan delivered by *MBVF* is in the worst case equal to the cost of the plan produced by *BVF*.

Note that lemma 2 does not imply that the behavior of *MBVF* is monotonic. It is possible that the cost of the plan derived by *MBVF* increases when more materialized views are available, but still it will be less or equal to the cost of *BVF's* plan.

5 Experimental Evaluation

In order to test the behavior of our algorithms under realistic conditions, we constructed three families of synthetic query sets larger than *q2_2*. Each query set contains 100 MDX queries. An MDX query can be analyzed into *S* sets of $|Q_{SET}|$ related SQL group-by queries. We generated the query sets as follows: For each MDX query we have randomly chosen *S* nodes q_1, q_2, ..., q_S in the corresponding lattice. Then, for each q_i, $1 \leq i \leq S$, we randomly selected $|Q_{SET}|$ nodes in the sub-lattice which is rooted in q_i. We denote each query set as $q|Q_{SET}|\_S$. For instance, the set *q50_1* captures the case where an MDX expression contains only related queries, while in *q1_50* the group-by queries are totally random.

In the first set of experiments, we assume that all queries use hash based star join. Figure 4 presents the cost of the plan versus S_{max} (the cost was calculated using the cost model from section 2). *GG* and *GG-c* produced similar results and *Steiner-1* outperformed them in most cases, so we only include the later algorithm in our figures. The SYNTH dataset is not shown due to lack of space; however the results were similar. The first row refers to the *q50_1* query set which is very skewed. Therefore it is easy to identify sets of queries that share their star joins. *BVF* is worse than *Steiner-1* for small values of S_{max} (i.e. small number of materialized views), but when S_{max} increases *Steiner-1* goes into the hard-region and its performance deteriorates. There are cases where the cost of its solution is higher that the *Top_Only* case (i.e. when there are no materialized views). *BVF* on the other hand, doesn't



(a) TPC-H dataset **(b)** APB dataset

Fig. 4. Total execution cost versus S_{max}. All queries use hash based star join. The first row refers to the *q50_1* query set and the second to the *q1_50* query set

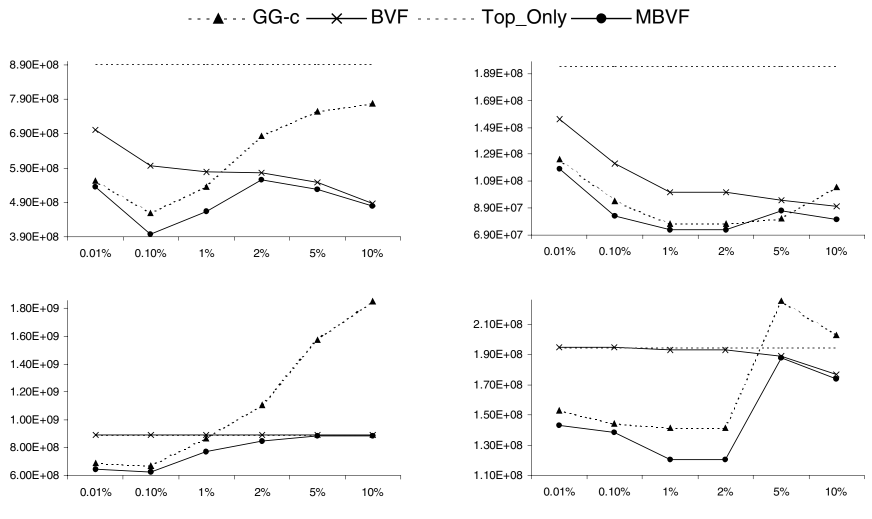
suffer from the hard-region problem, due to its monotonic property, so it is always better that the *Top_Only* case, and outperforms *Steiner-1* when S_{max} increases

MBVF was found to be better in all cases. For small values of S_{max} the algorithm is almost identical to *Steiner-1*, but when the later goes into the hard-region, *MBVF* follows the trend of *BVF*. Observe that *MBVF* is not monotonic. However, since it is bounded by *BVF*, it exits the hard region fast, and even inside the hard region, the cost of the plans does not increase dramatically.

In the second row of figure 4, we present the results for the *q1_50* query set. Although the trend is the same, observe that the cost of the plans of both *BVF* and *MBVF* approach the cost of the *Top_Only* plan. The reason is that the group-by queries inside *q1_50* are random, so there is a small probability that there exist many sets of related queries. Therefore, *BVF* and *MBVF* tend to construct plans with one or two sets and assign it to a general view, or even to the base tables. Similar results were obtained for the *q25_2* query set.

In the next experiment, we tested the general case where some of the group-by queries of each MDX are processed by hash-based star join, while the rest use index-based hash join. We run experiments where the percentage of the group-by queries that could use index-based star join was set to 50%, 25% and 10%. The subset of queries that could use the indices was randomly selected from our previous query sets. We did not consider the quite unrealistic case where all the queries would benefit from the indices. Our results suggest that the trend in all the tested cases was the same. In figure 5 we present the results for the 25% case (only *GG-c* is presented in the diagrams, since it delivered the best plans).

The results are similar with the case where only hash-based star joins are allowed. Observe however, that the distance of the produced plans from the *Top_Only* plan has



(a) TPC-H dataset **(b)** APB dataset

Fig. 5. Total execution cost versus S_{max}. 25% of the queries can use index based star join. The first row refers to the *q50_1* query set and the second to the *q1_50* query set.

increased in most cases. This is because the algorithms deliver plans that include shared index-based star joins, so they can achieve, in general, lower execution cost.

6 Conclusions

In this paper we conducted an extensive experimental study on the existing algorithms for optimizing multiple dimensional queries simultaneously in multidimensional databases, using realistic datasets. We concluded that the existing algorithms do not scale well if a set of views is materialized to accelerate the OLAP operations. Specifically, we identified the existence of a hard-region in the process of constructing an optimized execution plan, which appears when the number of materialized views increases. Inside the hard region the behavior of the algorithms is unstable, and the delivered plans that use materialized views can be worse than executing all queries from the base tables.

Motivated by this fact, we developed a novel greedy algorithm (*BVF*), which is monotonic and its worst-case performance is bounded by the case where no materialized views are available. Our algorithm outperforms the existing ones beyond the point that they enter the hard-region. However, *BVF* tends to deliver poor plans when the number of materialized views is small. As a solution, we developed a multilevel variation of *BVF*. *MBVF* is bounded by *BVF*, although it does not have the monotonic property. Our experiments indicate that for realistic workloads *MBVF* outperforms its competitors in most cases.

Acknowledgements. This work was supported by grants HKUST 6070/00E and HKUST 6090/99E from Hong Kong RGC.

References

[APB] OLAP Council, "OLAP Council APB-1 OLAP Benchmark RII", http://www.olapcouncil.org

[CCS93] Codd E.F., Codd S.B., Salley C.T., "Providing OLAP (on-line analytical processing) to user-analysts: An IT mandate", Technical report, 1993.

[GM99] Gupta H., Mumick I. S., "Selection of Views to Materialize Under a Maintenance-Time Constraint", Proc. ICDT, 1999.

[Gupt97] Gupta H., "Selection of Views to Materialize in a Data Warehouse", Proc. ICDT, 1997.

[HRU96] Harinarayan V., Rajaraman A., Ullman J. D., "Implementing data cubes efficiently", Proc. ACM-SIGMOD, 1996.

[KP00] Kalnis P., Papadias D., "Optimization algorithms for simultaneous multidimensional queries in OLAP environments", Technical report, HKUST-CS00-04, 2000, available at http://www.cs.ust.hk/~kalnis/hkust-cs00-04.pdf.

[KR99] Kotidis Y., Roussopoulos N., "DynaMat: A Dynamic View Management System for Data Warehouses", Proc. ACM-SIGMOD, 1999.

[LOY00] Liang W., Orlowska M. E., Yu, J. X., "Optimizing multiple dimensional queries simultaneously in multidimensional databases", The VLDB Journal, 8, 2000.

[MS] Microsoft Corp., "OLE DB for OLAP Design Specification", http://www.microsoft.com

[OQ97] O'Neil P., Quass D., "Improved query performance with variant indexes", Proc. ACM-SIGMOD, 1997.

[PS88] Park J., Segev A., "Using common subexpressions to optimize multiple queries", Proc. ICDE, 1988.

[RSSB00] Roy P., Seshadri S., Sudarshan S., Bhobe S., "Efficient and Extensible Algorithms for Multi Query Optimization", Proc. ACM-SIGMOD, 2000.

[S88] Sellis T. K., "Multi-query optimization", ACM Trans. On Database Systems, 13(1), 1988.

[SDN98] Shukla A., Deshpande P., Naughton J. F., "Materialized View Selection for Multidimensional Datasets", Proc. VLDB, 1998.

[SS94] Shim K., Sellis T. K., "Improvements on heuristic algorithm for multi-query optimization", Data and Knowledge Engineering, 12(2), 1994.

[Su96] Sundaresan P.: "Data warehousing features in Informix Online XPS", Proc. 4th PDIS, 1996.

[TPC] Transaction Processing Performance Council, "TPC-H Benchmark Specification", v. 1.2.1, http://www.tpc.org

[Zeli97] Zelikovsky A., "A series of approximation algorithms for the acyclic directed Steiner tree problem", Algorithmica 18, 1997.

[ZDNS98] Zhao Y., Deshpande P. M., Naughton J. F., Shukla A., "Simultaneous optimization and evaluation of multiple dimension queries", Proc. ACM-SIGMOD, 1998.

Improving the Performance of OLAP Queries Using Families of Statistics Trees

Joachim Hammer and Lixin Fu

Dept. of CISE, University of Florida, Gainesville, Florida 32611-6120, U.S.A.
{jhammer,lfu}@cise.ufl.edu

Abstract. We present a novel approach to speeding up the evaluation of OLAP queries that return aggregates over dimensions containing hierarchies. Our approach is based on our previous version of CubiST (Cubing with Statistics Trees), which pre-computes and stores all possible aggregate views in the leaves of a statistics tree during a one-time scan of the data. However, it uses a single statistics tree to answer all possible OLAP queries. Our new version remedies this limitation by materializing a family of derived trees from the single statistics tree. Given an input query, our new query evaluation algorithm selects the smallest tree in the family which can provide the answer. Our experiments have shown drastic reductions in processing times compared with the original CubiST as well as existing ROLAP and MOLAP systems.

1 Introduction

Online analytical processing (OLAP) [2,12] is a recent technology that enables decision support for large enterprise data warehouses. Starting point for OLAP is the data cube, a multi-dimensional organization of the underlying data, which is well suited for the analytical and navigational activities across the dimensions, hierarchies, and associated measures. However, on any but the smallest databases, OLAP queries tax even the most sophisticated query processors since the size of the data cube often grows to hundreds of GBs or TBs and the data is of high dimensionality with large domain sizes.

In [3], Fu and Hammer describe a new, efficient algorithm called CubiST (Cubing with Statistics Trees) for evaluating ad-hoc OLAP queries using so-called Statistics Trees (STs). In general, a statistics tree is a space-efficient encoding for all possible aggregates in the cube, and can be used to evaluate queries that return aggregate values (e.g., What is the percent difference in car sales between the southeast and northeast from 1990 until 2000?) rather than sets of tuples (e.g., Find the names and addresses of car dealers in the southeast who sold more cars in 1999 than in 1998). We have termed the former type of query *cube query*. However, the original version of CubiST does not efficiently support hierarchies, which are the result of partitioning dimension values into different levels of granularity (e.g., days into weeks, quarters, and years).

In this paper, we present a new methodology called CubiST^{++} to evaluate cube queries, which greatly improves upon the usefulness and efficiency of the original CubiST. CubiST^{++} specifically improves the evaluation of cube queries involving dimension hierarchies in a data cube. Instead of using a single (base) statistics tree to compute all cube queries, our new algorithms materialize a family of trees from which CubiST^{++} selects potentially smaller *derived* trees to compute the answer.

In general, research aimed at speeding up OLAP queries falls into the following categories: *Special-purpose OLAP servers* including Relational OLAP (e.g., MicroStrategy [11], Redbrick [15]) and Multi-dimensional OLAP (e.g., Arbor Systems Essbase [1], Oracle Express [14]), pre-computation of query fragments using *materialized views* (see for example, [6, 8, 9] for approaches to view selection as well as [10] for a collection of excellent papers on view materialization). Also relevant is the work on *index structures* (see [13] for an excellent overview) and *processing of aggregation queries* [5, 16]. However, to be able to support a wide range of OLAP queries, indexing and pre-computation of results alone will not produce good results. For example, building an index for each attribute of the warehouse or pre-computing every sub-cube requires too much space and results in unacceptable maintenance costs. We believe that our approach which is based on a new encoding for aggregates, a greedy strategy for selecting views, and a query processing algorithm represents a major step towards supporting ad-hoc OLAP queries efficiently.

2 Families of Statistics Trees

2.1 The Statistics Tree Data Structure

We start by briefly introducing the *statistics tree* (ST) data structure and corresponding algorithms to select and generate families of statistics trees. An ST tree is a multi-way, balanced tree whose leave nodes contain the aggregates for a set of records consisting of one or more attributes. By aggregates, we are referring to the result of applying an aggregation function (e.g., `count`, `min`, `max`) to each combination of dimension values in the data set. Leaf nodes are linked to facilitate retrieval of multiple aggregates. Each level in the tree (except the leaf level) corresponds to an attribute. An internal node has one pointer for each domain value, and an additional "star" pointer representing the entire attribute domain (the star has the intuitive meaning "any" analogously to the star in Gray's cube operator [4]). Internal nodes contain only tree node pointers. Initially, the values in the leave nodes are set to zero or undefined. Populating the tree is done during a one-time scan of the data set: for each record, the tree is traversed based on the dimension values in the record; the leaf nodes at the end of the path are updated using the aggregation function (e.g., incremented by one in the case of the `count` function). In a sense, an ST can be considered a superview since it stores multiple traditional views representing different group-by combinations. However, it differs from multi-dimensional arrays in that it combines all arrays into a single structure which simplifies initialisation and maintenance especially when the number of dimensions is not known in advance or changes over time.

Fig. 1 depicts a simple statistics tree corresponding to a data cube with three dimensions having cardinalities d_1=2, d_2=3, and d_3=4 respectively. The contents of the tree are shown after inserting the record (1,2,4)[1] into an empty tree. The leaf nodes

[1] For efficiency, the values "1", "2", and "4" are integer representations for the domain values of the three dimensions. For example, if the first dimension represents `Manufacturer`, "1" may represent the value "Ford", etc. The mappings for the other two dimensions work similarly.

store the result of applying the aggregation function `count` to the data set. The update paths relating to the sample record are shown as solid thick lines in Fig. 1. The amount of memory that is needed to store a k-dimensional statistics tree is bounded by $c(d_1+1)(d_2+1)...(d_k+1)$, where d_i is the cardinality of dimension i, and the constant value c accounts for the space that is needed to store the internal nodes of the tree.

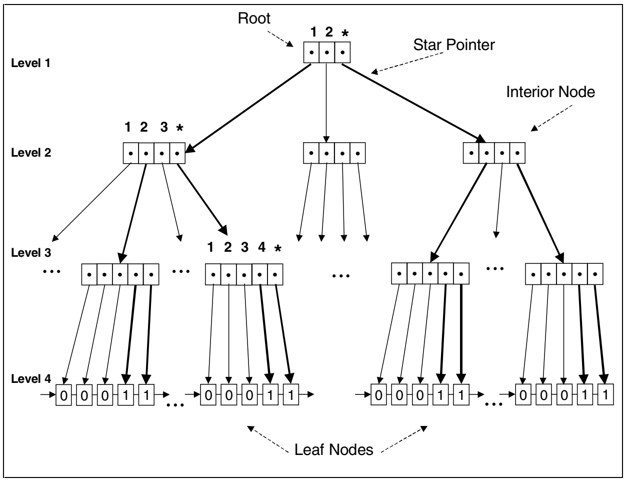


Fig. 1. Statistics tree after processing input record (1,2,4)

The original CubiST provides a useful framework for answering cube queries. However, it does not efficiently answer queries on data cubes containing *hierarchies* on the dimensional values (e.g., day, month and year in the case of the time dimension) since it only stores data at the finest level of granularity (e.g., time in terms of days). In those cases, one must first transform the conditions of the query, which may reference different levels of the hierarchy, into conditions involving the level of granularity represented by the ST (i.e., days), and then treat them as range queries or partial queries over the detailed data. This is particularly inefficient for queries involving only conditions on the coarsest level of granularity which could have been answered using a much smaller ST. One way to reduce the I/O cost for the large STs is to transmit only the leaves; the internal nodes can be generated in memory without additional I/O. A more effective approach, and the one described in this paper, is to materialize additional smaller trees, each representing the dimensions at one particular level of the hierarchy. In most cases, cube queries can be answered using one of the smaller trees rather than the single large ST as is done in CubiST. We term this single large ST as defined in the original version of CubiST *base tree*. Using the base tree, one can compute and materialize other statistics trees with the same dimensions but containing different hierarchy levels for one or more of the dimensions. We term these smaller trees *derived trees*. A base tree and its derived trees form a *family of statistics trees* with respect to the base tree. Accordingly, we call the new algorithm CubiST^{++} (CubiST plus families). Before presenting its details, we first describe how to represent hierarchies.

2.2 Hierarchical Partitioning and Mapping of Domain Values

In relational systems, dimensions containing hierarchies are stored in a row-column format in which different levels are represented as separate attributes. As before, we map the domain values into integers. However, in CubiST^{++} we also include the hierarchies in our encoding: a value at a higher level of abstraction includes multiple values at a lower level. By scanning a dimension table, we can establish the domain value mappings, hierarchical partitions, as well as the inclusion relationships.

Consider Table 1, which represents the location information in a data source. We first group the records by columns `Region`, `State` and `City`, remove duplicates, and then map the domain values into integers using the sorted order. The result is shown in Table 2 (numbers in parentheses denote the integer representations). For example, value "1" for the `Region` dimension ("MW") includes values "1" and "2" for the `State` dimension ("IL", "MN") as well as values "1" and "2" for the `City` dimension ("Chicago", "Twin City").

Table 1. Original location data

LocationID	*City*	*State*	*Region*
1	Gainesville	FL	SE
2	Atlanta	GA	SE
3	Los Angeles	CA	WC
4	Chicago	IL	MW
5	Miami	FL	SE
6	San Jose	CA	WC
7	Seattle	WA	WC
8	Twin City	MN	MW

Table 2. Partitioning and mapping

Region	*State*	*City*
MW (1)	IL (1)	Chicago (1)
	MN (2)	Twin City (2)
SE (2)	FL (3)	Gainesville (3)
		Miami (4)
	GA (4)	Atlanta (5)
WC (3)	CA (5)	Los Angeles (6)
		San Jose (7)
	WA (6)	Seattle (8)

2.3 A Greedy Algorithm for Selecting Family Members

To generate a family of statistics tree, we first need to choose from the set of all possible candidate trees (i.e., trees containing all combinations of dimensions and hierarchy levels) the best set of candidate trees. Second, we need to compute the selected trees.

Potentially, any combination of hierarchy levels can be selected as a family member. However, we must consider space limitations and maintenance overhead. We introduce a greedy algorithm to choose the members in a systematic fashion. Starting from the base tree, during each step, we roll-up the values of the largest dimension to its next level in the hierarchy, keeping other dimensions unchanged. This newly formed ST becomes a new member of the family. The iteration continues until each dimension is rolled-up to its coarsest level of granularity or until the size of the family exceeds the maximum available space. The total size of the family is less than the single base tree.

Suppose a base tree T_0 represents a data set with three dimensions d_1, d_2, d_3 whose cardinalities are 100 each. Each dimension is organized into ten groups of ten elements creating a two-level hierarchy. We use the superscripted hash mark '#' to indicate values corresponding to the second hierarchy level. For example, $0^{\#}$

represents values 0 through 9 from the first hierarchy level, $1^\#$ represents values 10 through 19, etc. The degrees of the internal nodes of T_0 are 101 for each level of the tree (the extra 1 accounts for the star pointer). The selected derived trees T_1, T_2, T_3 that make up a possible family together with T_0 are as follows: T_1 is the result of rolling up d_2 in T_0 (since all three dimensions are of the same size, we pick one at random). T_1 has degrees 101, 101 and 11. T_2 is the result of rolling up d_1 in T_1. Its degrees are 101, 11, and 11. T_3 is the result of rolling up d_0 in T_2. Its degrees are 11, 11, and 11.

2.4 Deriving a New Statistics Tree

Except for the base tree, which is computed from the initial data set, each derived tree is computed by rolling-up one of its dimensions. Before introducing our new tree derivation algorithm, we first define the a new ST merge operator "⊕" as follows:

Definition 2.1: Two STs are *isomorphic* if they have the same structure except for the values of their leaves. *Merging* two isomorphic ST's *S1, S2* results in a new ST *S* that has the same structure as *S1* or *S2* but its leaf values are the sum of the corresponding leaf values of *S1* and *S2*. This relationship is denoted by $S=S_1 \oplus S_2$.

In order to derive a new tree, the derivation algorithm proceeds from the root to the level (i.e., the dimension) that is being rolled up. Here, we reorganize and merge the sub-trees for all the nodes in that level to form new subtrees. For each node, we adjust the degree and update its pointers to reference the newly created subtrees.

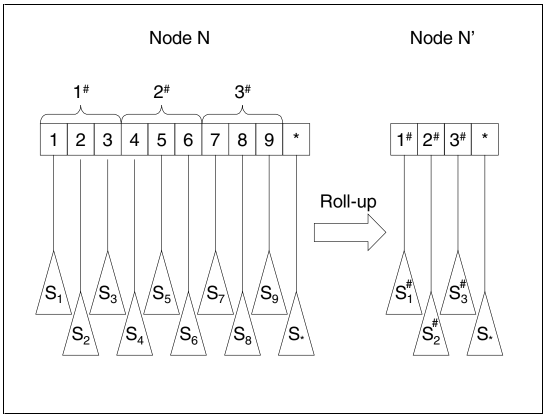


Fig. 2. Rolling-up the dimension values in a two-level hierarchy

The best way to illustrate the derivation algorithm is through an example. Suppose that during the derivation, we need to roll-up a dimension that has nine values and forms a two-level hierarchy consisting of three groups with three values each. A sample node *N* of this dimension with children S_1, ..., S_9, S_* is shown on the left of Fig. 2. We merge its nine subtrees into three new subtrees $S_1^\#$, $S_2^\#$ and $S_3^\#$ each having three subtrees. The new node *N'* with degree four is shown on the right side of the figure. The star subtree remains the same. Using the definition from above, we can see that the following relationships hold among the subtrees of nodes *N* and *N'*:

$$S_1^\#=S_1 \oplus S_2 \oplus S_3,\ S_2^\#=S_4 \oplus S_5 \oplus S_6, \text{ and } S_3^\#=S_7 \oplus S_8 \oplus S_9 .$$

3 Answering Cube Queries Using a Family of Statistics Trees

Once a family is created, it can be used to answer cube queries. We have designed a new language called *Cube Query Language (CQL)* for representing cube queries efficiently. After introducing CQL, we will describe our query processing strategy.

3.1 The Cube Query Language CQL

A cube query contains the following information: the desired measure(s), the aggregate operation used, the dimensions and their hierarchy levels with constraints, and the selected value subsets of the domains. CQL represents cube queries as follows:

`AGGR-OPERATOR`$^{\text{MEASURES}}$`((D-INDEX,H-LEVEL):SELECTED-VALUES; …).`

Constraints, which appear in the body of the query between the parentheses, are separated by semicolons. Each constraint contains the dimension (D-INDEX) as well as the hierarchy level (H-LEVEL). The selected values can be specified in one of three ways: as a single value, a range, or a partial selection. If there is no constraint (null constraint) on a dimension, we can specify an "ALL" value for it. The hierarchy level can be omitted if the constrained dimension is non-hierarchical or the selected values are at the lowest level of the dimension (default value 0). In this case, the constraint is simplified as D-INDEX: SELECTED VALUES.

The following is a CQL representation of the cube query "Find the total (car) sales for Orlando from March through October." For clarity, we are using actual names and values rather than the integer representations:

q=`SUM`$^{\text{SALES}}$`((Time,Month):[Mar,Oct];(Location,City):Orlando).`

3.2 Choosing the Optimal Derived Tree

Given a family of trees, more than one tree may be able to answer a given query; however, the smallest possible one is preferred. The desired tree is the tree which contains the highest level of abstraction that matches the query. This tree can be found using our query-matching scheme, which is based on the following definition.

Definition 3.1: A tree matrix (TM) is a matrix in which rows represent dimensions, columns represent the hierarchy levels, and the row/column intersections contain the labels of the ST's in the family. Intuitively, a TM describes the views that are materialized in the hierarchies.

Level of Abstraction / Dimension	*level 0*	*level 1*
1	T_0, T_1, T_2	T_3
2	T_0, T_1	T_2, T_3
3	T_0	T_1, T_2, T_3

Fig. 3. A sample tree matrix

Continuing with our sample family of trees from Sec. 2.3, T_1 has degrees 101, 101, 11. Its first two dimensions have not been rolled-up (level 0 of the hierarchy) whereas the third dimension has been rolled-up to level 1; hence T_1 is placed in column 1 for rows 1 and 2 and in column 2 for row 3. Trees T_0 and T_1 are placed accordingly as shown in Fig. 3. Next, we provide three definitions for representing hierarchies and for what constitutes an optimal match between a query and an ST.

Definition 3.2: A query hierarchy vector (QHV) is a vector $(l_1,l_2,...,l_k)$, where l_i is the i^{th} dimension level value in its hierarchy in the query.

Definition 3.3: A QHV $(l_1,l_2,...,l_k)$ matches view T in the TM iff $l_i \geq$ column index of the i^{th} row entry of T in TM, $\forall i$, i=1,2,...,k. In this case, T's *column index vector*, which is composed of T's column indices, is less than QHV.

Definition 3.4: An optimal matching tree with respect to a query is the tree in TM that has the largest index number and matches the query's QHV.

Consider the sample cube query `q=count((1,1):3,(2,0):2,(3,1):5)`. Its QHV is (1,0,1). We start by matching the smallest tree T_3. Since the column vector for T_3 is (1,1,1), which does not match (1,0,1), T_3 is not considered. For the same token, the second smallest tree T_2 is also not an optimal match. However, the column index vector of T_1 is (0,0,1) ≤ (1,0,1), i.e. T_1 matches QHV. Hence T_1 is the optimal matching tree which is used to answer *q*.

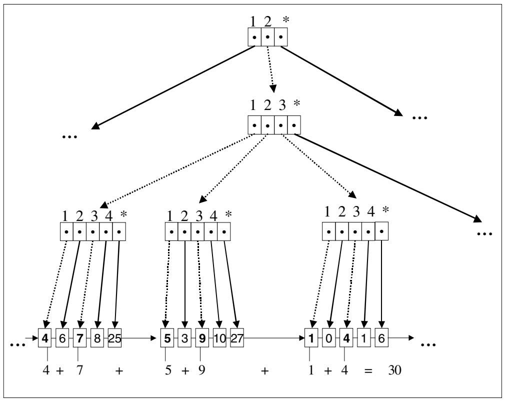


Fig. 4. Using T_1 to answer a cube query

3.3 Rewriting and Answering the Query

Each query is evaluated based on its optimal matching ST. Before evaluation, the query must be rewritten if its QHV is not equal to the ST's column index vector. For example, query `q=count((1,1):3,(2,0):2,(3,1):5)` must be rewritten into `q'=count([30,39],2,(3,1):5)` so that its new QHV (0,0,1) exactly equals the levels of optimal ST (in this case T_1).

Next, we compute the query using T_1 by applying the original query evaluation algorithm of CubiST. Due to space limitations, we can only highlight the query processing steps using an example; complete details can be found in [7]. Fig. 4 displays T_1 after initialisation. To answer query `q=count(2,[1,3],{1,3})`, we follow the second pointer of the root to the node of level 2 and then the first three pointers to the nodes of level 3. From there, we follow the first and third pointers to the leaves that contain the partial aggregates for `count` (shown in bold). The summation of the counts is the final answer. The paths taken by the algorithm are shown as dashed lines.

4 Performance Evaluation

We have evaluated CubiST^{++} in three different scenarios: against the original version which does not use families, against a popular index structure, and against a commercial RDBMS which has been fine-tuned for OLAP. In all three experiments, CubiST^{++} has shown promising performance. To conduct the experiments, we have simulated a relational warehouse using synthetic data sets. The data elements are uniformly distributed across the underlying domain for each column. For each set of experiments, a set of randomly generated queries is used. The testbed consist of a SUN ULTRA 10 workstation running Sun OS 5.6 with 90MB of RAM.

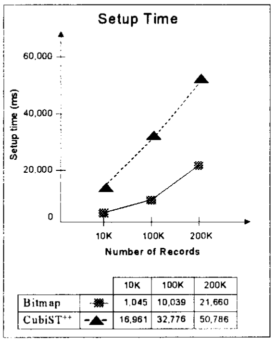
In the first experiment, we show the effectiveness of using families of statistics trees. We have fixed the number of records to *r*=1,000,000. We also assume there are three dimensions, each of which has a two-level hierarchy. To simplify the hierarchical inclusion relationships, we further assume that each second level value includes an equal number of first level values; this number is referred to as *factor*. The test query is `q=count(`s_1`)=count([0,29])`. For both approaches, we compare the size of the ST's (based on number of leaves), the I/O time spent on loading the ST into memory, and the total query answering time. Table 3 summarizes the results, which show a dramatic decrease in I/O and query processing time in favour of CubiST^{++}.

Table 3. Sample running times for CubiST^{++} using different derived trees

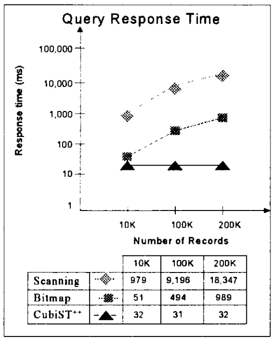
Factors	1, 1, 1	3, 2, 2	4, 4, 3
Dimension Sizes	60, 60, 60	20, 30, 30	15, 15, 20
Number of Leaves	226,981	20,181	5,376
I/O Time (ms)	16,818	1,517	384
Total Time (ms)	25,387	1,655	446

In the second experiment, we compare the setup and response times of CubiST^{++} with those of two other approaches: A naive query evaluation technique henceforth referred to as *scanning* and a bitmap-based query evaluation algorithm referred to as *bitmap*. The bitmap-based query evaluation algorithm uses a bitmap index structure to answer cube queries without touching the original data set when the bit vectors are already setup. In this series of experiments, the domain sizes of the five dimensions in the data set are 10, 10, 15, 15, and 15 respectively.

Fig. 5 shows the results for the query `q=count([2,8],[2,8],[5,10])`. Although the setup time for CubiST^{++} is larger than that of Bitmap, its response time is faster and independent of the number of records.



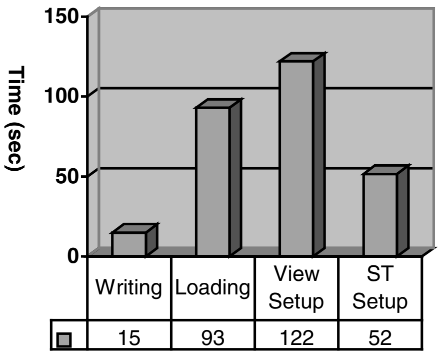
		10K	100K	200K
Bitmap	▨	1,045	10,039	21,660
CubiST++	▲	16,961	32,776	50,786



		10K	100K	200K
Scanning	◈	979	9,196	18,347
Bitmap	▨	51	494	989
CubiST++	▲	32	31	32

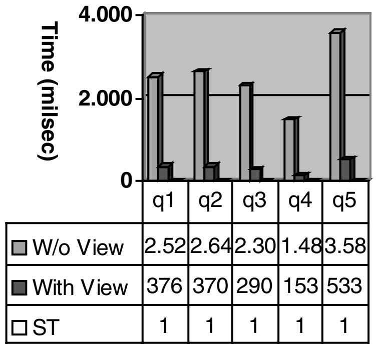
Fig. 5. CubiST^{++} vs. Bitmap: setup and response times vs. number of records

In our last set of experiments, we compare running times of CubiST^{++} against those of a commercial RDBMS. The single relational table in the warehouse has 1,000,000 rows and 5 columns with domain sizes 10, 10, 10, 10, and 15. First, we loaded the table and answered five randomly generated cube queries using the commercial ROLAP engine. Next, we created a view to group all five columns in support of the `count` operation. The view was materialized and used to answer the same set of queries. The same queries were also processed by CubiST^{++}.



	Writing	Loading	View Setup	ST Setup
□	15	93	122	52

Fig. 6. Setup times



	q1	q2	q3	q4	q5
□ W/o View	2.52	2.64	2.30	1.48	3.58
■ With View	376	370	290	153	533
□ ST	1	1	1	1	1

Fig. 7. Query response times

The results are shown in Figures 6 and 7. The column labelled "Writing" in Figure 6 refers to the time it takes to write the entire table to disk (for comparison only). Notice that in Figure 7 the query execution times for CubiST^{++} are around 1ms and cannot be properly displayed using the relatively large scale. It is worth pointing out that the evaluation times of CubiST^{++} are two orders of magnitude faster than those posted by the RDBMS using the materialized view.

5 Conclusions

In this paper, we have presented an overview of CubiST^{++}, a new algorithm for efficiently answering cube queries that involve constraints on arbitrary abstraction hierarchies. Using a single base statistics tree, we select and derive an appropriate family of statistics trees which are smaller then the base tree and, when taken together, can answer most ad-hoc cube queries while reducing the requirements on I/O and memory. We have proposed a new greedy algorithm to choose family members and a new roll-up algorithm to compute derived trees. Using our matching algorithm, the smallest tree that can answer a given query is chosen. Our experiments demonstrate the significant performance benefits of CubiST^{++} over existing OLAP techniques.

References

1. Arbor Systems, "Large-Scale Data Warehousing Using Hyperion Essbase OLAP Technology," Arbor Systems, White Paper, `www.hyperion.com/whitepapers.cfm`
2. S. Chaudhuri and U. Dayal, "An Overview of Data Warehousing and OLAP Technology," *SIGMOD Record*, **26**:1, pp. 65-74, 1997
3. L. Fu and J. Hammer, "CubiST: A New Algorithm for Improving the Performance of Ad-hoc OLAP Queries," *Proceedings of the ACM Third International Workshop on Data Warehousing and OLAP (DOLAP)*, Washington, DC, pp. 72-79, 2000
4. 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*, **1**:1, pp. 29-53, 1997
5. A. Gupta, V. Harinarayan, and D. Quass, "Aggregate-query Processing in Data Warehousing Environments," *Proceedings of the Eighth International Conference on Very Large Databases*, Zurich, Switzerland, pp. 358-369, 1995
6. H. Gupta and I. Mumick, "Selection of Views to Materialize Under a Maintenance Cost Constraint," Stanford University, Technical Report
7. J. Hammer and L. Fu, "Speeding Up Data Cube Queries with Statistics Trees," University of Florida, Gainesville, FL, Research report TR01-007, January 2001
8. W. Labio, D. Quass, and B. Adelberg, "Physical Database Design for Data Warehouses," in *Proceedings of the International Conference on Database Engineering*, Birmingham, England, pp. 277-288, 1997
9. M. Lee and J. Hammer, "Speeding Up Warehouse Physical Design Using A Randomized Algorithm," *Proceedings of the International Workshop on Design and Management of Data Warehouses (DMDW '99)*, Heidelberg, Germany, 1999
10. D. Lomet, ed. *Bulletin of the Technical Committee on Data Engineering*. Special Issue on Materialized Views and Data Warehousing, ed. J. Widom. **18**, IEEE Computer Society, 1995
11. MicroStrategy Inc., "The Case For Relational OLAP," MicroStrategy, White Paper, `www.microstrategy.com/publications/whitepapers/Case4Rolap/`
12. OLAP Council, "OLAP Overview and Definitions," `www.olapcouncil.org/`
13. P. O'Neil and D. Quass, "Improved Query Performance with Variant Indexes," *SIGMOD Record*, **26**:2, pp. 38-49, 1997
14. Oracle Corp., "Oracle Express OLAP Technology," `www.oracle.com/olap/`
15. Redbrick Systems, "Informix Redbrick Decision Server," Redbrick, Los Gatos, CA, Product Overview and Data Sheet, `www.informix.com/redbrick/`
16. W. P. Yan and P. Larson, "Eager Aggregation and Lazy Aggregation," *Proceedings of the Eighth International Conference on Very Large Databases*, Zurich, Switzerland, pp. 345-357, 1995

Changes of Dimension Data in Temporal Data Warehouses

Johann Eder and Christian Koncilia

University of Klagenfurt
Dep. of Informatics-Systems
{eder,koncilia}@isys.uni-klu.ac.at

Abstract. Time is one of the dimensions we frequently find in data warehouses allowing comparisons of data in different periods. In current multi-dimensional data warehouse technology changes of dimension data cannot be represented adequately since all dimensions are (implicitly) considered as orthogonal. We propose an extension of the multi-dimensional data model employed in data warehouses allowing to cope correctly with changes in dimension data: a temporal multi-dimensional data model allows the registration of temporal versions of dimension data. Mappings are provided to transfer data between different temporal versions of the instances of dimensions and enable the system to correctly answer queries spanning multiple periods and thus different versions of dimension data.

1 Introduction and Motivation

Data warehouses or data marts are integrated materialized views over several often heterogeneous data sources. Their most important usage is On-Line Analytical Processing (OLAP) typically using a multi-dimensional view of the data. OLAP tools then allow to aggregate and compare data along dimensions relevant to the application domain. Typical examples of dimensions found frequently in business data warehouses include time, organizational structure (divisions, departments, etc.), space (cities, regions, countries) and product data.

This multi-dimensional view provides long term data that can be analyzed along the time axis, in contrast to snapshot-based OLTP systems. Available OLAP systems are therefore prepared to deal with changing values of fact data, e. g., changing profit or turnover but surprisingly not for modifications in dimension data, e. g., if a new branch or division is established, although time is usually explicitly represented as a dimension in data warehouses.

The reason for this disturbing property of current data warehouse technology is the implicitly underlying assumptions that the dimensions are orthogonal. Orthogonality with respect to the dimension time means the other dimensions ought to be time-invariant. This silent assumptions inhibits the proper treatment of changes in dimension data.

Naturally, it is vital for the correctness of results of OLAP queries that modifications of dimension data is correctly taken into account. E. g., when the economic figures of European countries over the last 20 years are compared on a country level, it is essential to be aware of the re-unification of Germany, the separation of Czechoslovakia, etc. Business structures and even structures in public administration are nowadays subject to highly dynamic changes. Comparisons of data over several periods, computation of trends, etc. have the necessity to correctly and adequately treat changes in dimension data. Otherwise we face meaningless figures and wrong conclusions triggering bad decisions. From our experience we could cite too much such cases.

The following extensions to a data warehouse are therefore necessary:

- **Temporal extension**: dimension data has to be time stamped in order to represent their valid time.
- **Structure versions**: by providing time stamps for dimension data the need arises that our system is able to cope with different versions of structures.
- **Transformation functions**: Our system has to support functions to transform data from one structure version into another.

In contrast to temporal databases, which have been well studied, e. g., [3,7, 8], few approaches are known in literature for temporal data warehouses, e. g., [4,13]. The same holds for schema evolution of databases, e. g., [11,5] vs. schema evolution of data warehouses, e. g., [2].

[4] present necessary extensions to a data warehouse to cover temporal aspects, in particular to keep track of the history of hierarchical assignments. [2,1] deal with schema evolution and schema versioning for data warehouse systems, transfering changes of the conceptual schema can be automatically into the logical and internal schema. However, these papers do not address the inevitable consequences of these scheme evolutions for analytical queries.

A formal definition of a temporal OLAP system and a temporal query language (TOLAP) is proposed in [12], however, without transformation of data between structural versions the system is not able to cope with changes in the time and fact dimensions.

In another proposal [13], the schema is extended with time stamps to enable the user to analyze data for different scenarios. However, this approach is limited to some basic operations on dimension data (e. g., insert/delete a dimension member; change the "parent" of dimension member).

2 Temporal Multidimensional Systems

A multidimensional view on data consists of a set of dimensions. defining an n-dimensional data cube [14,10]. Usually, a data cube is defined by a dimension *Time*, a dimension *Facts* and by several dimensions describing the managerial structures such as divisions, products, or branches.

A dimension is a set of dimension members and their hierarchical structure. For example *"VCR X-25"*, *"VCRs"* and *"All Products"* are dimension members of the dimension "Products" and are in the hierarchical relation *All Products* $\rightarrow$ *VCRs* $\rightarrow$ *VCR X-25*. The hierarchical structure of all dimensions defines all possible consolidation paths, i. e., it defines all possible aggregation and disaggregation paths.

We will now extend this description of a multi-dimensional system to define a temporal data warehouse supporting valid time relations:

- **Chronons**: A chronon Q is defined as "a non-decomposable time interval of some fixed, minimal duration" [9]. This means a chronon is the finest dimension member in the dimension time and the time axis defined through the dimension time is a series of chronons.
- **Time Intervals**: All dimension members and all hierarchical links between these dimension members are associated with a time interval $[T_s, T_e]$ representing the valid time beginning at T_s and ending at T_e (with $T_e \geq T_s$).

More formally, a temporal multidimensional system consists of:

i.) A number of dimensions $N + 1$.

ii.) A set of dimensions $\mathcal{D} = \{D_1, ..., D_N, F\}$ where F is the dimension describing the required facts and D_i are all other dimensions including a time dimension if required.

iii.) A number of dimension members M.

iv.) A set of dimension members $\mathcal{DM} = \mathcal{DM}_{D_1} \cup ... \cup \mathcal{DM}_{D_N} \cup \mathcal{DM}_F = \{DM_1, ..., DM_M\}$ where $\mathcal{DM}_F$ is the set of all facts, $\mathcal{DM}_{D_i}$ is the set of all dimension members which belong to dimension D_i. A dimension member DM_i is defined as $DM_i =< DM_{id}, Key, D_i, \mathcal{UDA}, [T_s, T_e] >$. DM_{id} is a unique identifier for each dimension member that cannot be changed (similar to $O_{id's}$ in object-oriented database systems). $[T_s, T_e]$ represents the valid time of the dimension member. D_i is the dimension identifier to which the dimension member belongs. Key is a user defined key (e. g., the number of a product) which is unique within D_i for each timepoint $T_s \leq T \leq T_e$. $\mathcal{UDA}$ is a set of user defined attributes (e. g., the name and/or color of a product).

v.) A set of hierarchical assignments $\mathcal{H} = \{H_1, ..., H_O\}$ where $H_i =< DM^C_{id}, DM^P_{id}, Level, [T_s, T_e] >$. DM^C_{id} is the identifier of a dimension member, DM^P_{id} is the dimension member identifier of the parent of DM^C_{id} or $\emptyset$ if the dimension member is a top-level dimension member. $Level$ is a value $0...L$ where L is the number of layers and $Level$ is the level of DM^C_{id}. All "leaves" (dimension members without successors) are at level 0. $[T_s, T_e]$ is the time stamp representing the valid time for the relation between DM^C_{id} and DM^P_{id}. No dimension member may be its own parent/child and cycles within $\mathcal{H}$ are not admissible.

vi.) A function $cval : (DM_{D_1}, ..., DM_{D_N}, DM_F) \rightarrow value$ which uniquely assigns a value to each vector $(DM_{D_1}, ..., DM_{D_N}, DM_F)$ where $(DM_{D_1}, ..., DM_{D_N}, DM_F) \in \mathcal{DM}_{D_1} \times ... \times \mathcal{DM}_{D_N} \times \mathcal{DM}_F$. Therefore, a cube (or n-cube) C is defined by this function $cval$. The domain of this cube $dom(C)$ is the set of all cell references. The range of this cube $ran(C)$ are all cell values.

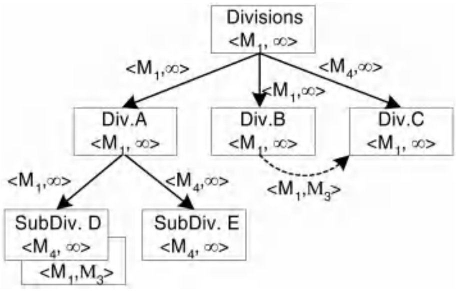


Fig. 1. A Dimension *Divisions* with time stamps

3 Structure Versions

Temporal projection and selection [9] allows us to define a *Structure Version* (SV) of a temporal data warehouse. Intuitively, a structure version is a view on a multidimensional structure that is valid for a given time interval $[T_s, T_e]$. All dimension members and all hierarchical relations are also valid for the given time interval. In other words: within one structure version no dimension member is changed nor a hierarchical relation. Vice versa each modification of a dimension member or a hierarchical relation leads to a new structure version.

Formally, each structure version is a 4-tuple $< SV_{id}, T, \{\mathcal{DM}_{D_1,SV_{id}}, ..., \mathcal{DM}_{D_N,SV_{id}}, \mathcal{DM}_{F,SV_{id}}\}, \mathcal{H}_{SV_{id}} >$ where SV_{id} is a unique identifier and T represent the valid time of that structure version as a time interval $[T_s, T_e]$. $\mathcal{DM}_{D_i,SV_{id}}$ ($\mathcal{DM}_{D_i,SV_{id}} \subseteq \mathcal{DM}_{D_i}$) is a set of all dimension members which belong to dimension D_i and which are valid at each timepoint P with $T_s \leq P \leq T_e$. $\mathcal{DM}_{F,SV_{id}}$ ($\mathcal{DM}_{F,SV_{id}} \subseteq \mathcal{DM}_F$) is the set of all facts which are valid at each timepoint P with $T_s \leq P \leq T_e$. $\mathcal{H}_{SV_{id}}$ ($\mathcal{H}_{SV_{id}} \subseteq \mathcal{H}$) is a set of hierarchical assignments valid at each timepoint P with $T_s \leq P \leq T_e$.

Conceptually each structure version SV has a corresponding cube with the same valid time interval. Fig. 1 shows an example for the consolidation tree of the dimension "Divisions" including time intervals. Each node and each edge in this figure has a time interval $[T_s, T_e]$. An attribute of "*SubDiv.D*" was modified at M_4, a new subdivision "*SubDiv.E*" was introduced at M_4 and *Div.C* was a subdivision of *Div.B* from M_1 until M_3 (dotted line). Two structure versions can be identified in this example:

i.) $<SV_1, [M_1, M_3], \{\{Divisions, Div.A, Div.B, Div.C, SubDiv.D\}, \{Sales\}\}, \{Div.A \rightarrow Divisions, SubDiv.D \rightarrow Div.A, ...\} >$
ii.) $<SV_2, [M_4, \infty], \{\{Divisions, Div.A, Div.B, Div.C, SubDiv.D, SubDiv.E\}, \{Sales\}\}, \{Div.A \rightarrow Divisions, SubDiv.D \rightarrow Div.A, ...\} >$.

In this example we have two different structure versions SV_1 and SV_2. SV_1 and all given dimension members (*Divisions*, *Div.A*, *Div.B*, ...) and hierarchical assignments (*Div.A* $\rightarrow$ *Divisions*, ...) are valid from M_1 to M_3. SV_2 is valid from M_4 to ∞, i. e., until now.

4 Structural Changes

We define the structure of a data warehouse (DWH) as a non-empty, finite set of structure versions DWH $= \{SV_1, ..., SV_n\}$, where each structure version SV_i is a 4-tuple $< SV_{id}, T, \mathcal{DM}_{SV_{id}}, \mathcal{H}_{SV_{id}} >$ (see above) forming a dense sequence of tuples $< SV_{id}$, T_i, $\mathcal{DM}_{SV_{id}}$, $\mathcal{H}_{SV_{id}} >$ with respect to chronon Q, i. e. , $T_i = [T_{i,s}, T_{i,e}]$ such that $T_{i,s} = T_{(i-1),e} + Q$.

We provide three basic operations `INSERT`, `UPDATE` and `DELETE` to modify the structure of a temporal data warehouse, i. e. , the dimension data within the granularity defined through the chronon Q. Key, D_i, $\mathcal{UDA}$ and DM^P_{id} are defined as described in Sect. 2

`INSERT(`DM`, `T_s`):` inserts the new dimension member DM as $<$ Key, D_i, $\mathcal{UDA}$, $DM^P_{id} >$. T_s defines that DM is valid for the time interval $[T_s, \infty]$. A unique DM_{id} is assigned to the new element.

`UPDATE(`Key`,`D_i`, `DM'`, `T_s`):` changes an existing dimension member identified by Key and D_i to a new dimension member DM' as $<$ Key, $\mathcal{UDA}$, $DM^P_{id} >$. An `UPDATE` operation consists of two actions: set the ending time of an existing dimension member to $T_s - Q$, and insert a new dimension member, with the valid time interval $[T_s, \infty]$.

`DELETE(`DM`, `T_e`):` changes the end time of the dimension member DM to T_e.

Using these basic operations we can modify the structural dimensions of the multidimensional cube. We distinguish among the following modifications:

i.) SPLIT: One dimension member splits into n dimension members.
ii.) MERGE: n dimension members are merged into one dimension member.
iii.) CHANGE: An attribute, e. g. the product number, of a dimension member changes.
iv.) MOVE: Modify the hierarchical position of a dimension member.
v.) NEW-MEMBER: Insert a new dimension member.
vi.) DELETE-MEMBER: Delete a dimension member.

5 Mappings between Structure Versions

We will now extend the temporal model of a data warehouse presented in chapter 2 with the definition of mapping functions between structure versions.

A structure version is a view on a temporal data warehouse valid for a given time period $[T_s, T_e]$. We distinguish between the structure of a structure version (the set of all valid dimension members of the structure version together with their hierarchies) and the data of a structure version (the cube defined by mapping the structure to the value domain).

For answering queries on the data warehouse the user always has to define which structure version should be used. The data returned by the query can, however, originate in several (different) temporal versions of the cube. Therefore, it is necessary to provide transformation functions mapping data from one structure version to a different structure version.

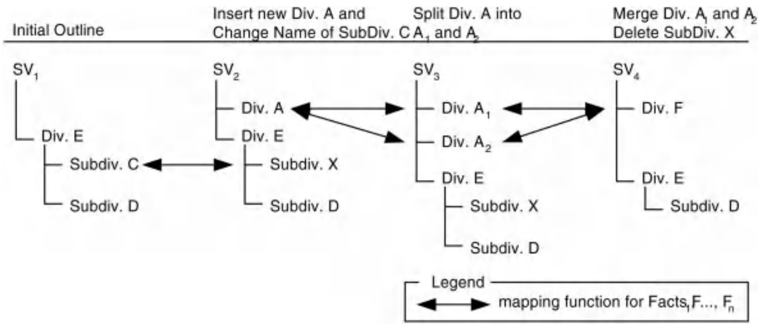

Initial Outline
Insert new Div. A and Change Name of SubDiv. C
Split Div. A into A1 and A2
Merge Div. A1 and A2 Delete SubDiv. X
SV1
Div. E
Subdiv. C
Subdiv. D
SV2
Div. A
Div. E
Subdiv. X
Subdiv. D
SV3
Div. A1
Div. A2
Div. E
Subdiv. X
Subdiv. D
SV4
Div. F
Div. E
Subdiv. D
Legend
mapping function for Facts F1..., Fn

Fig. 2. An example for structural changes and mapping functions

In the rest of the paper we make the following assumptions: Relations between different structure versions depend on the contemplated fact. For sake of simplicity and understandability we only consider the cube for a single fact. Furthermore, the cell values of upper-level dimension members are always computed from their subordinate lower level dimension members. Therefore, without loss of generality, we do not consider the upper levels here and assume that the dimensions are flat. Or, in other terms: before we transform the data we select the cube of the dimension members at level-0 and transform only this subset of cell values and compute the upper-levels of the resulting cube bottom-up as usual.

5.1 Definition of Inter-structure Relationships

Mapping functions are employed to map data (cell values) for numeric facts and a particular dimension member from one structure version into another using a weighting factor. $MapF$ is defined as $MapF(SV_j, SV_k, DM_{id}, DM'_{id}, \{M^1_{id}, ..., M^n_{id}\}, w)$ where SV_j and SV_k are different structure versions. DM_{id} and DM'_{id} are unique IDs for dimension members for which $DM_{id} \in SV_j$ and $DM'_{id} \in SV_k$ is true. DM_{id} and DM'_{id} must be dimension members of the same dimension. $\{M^1_{id}, ..., M^n_{id}\}$ is a non-empty, finite set of fact IDs and $\exists f : f \in F \wedge f_{id} = M^i_{id}$. w is the weighting factor to map data from one structure version into another.

We implicitly introduce a mapping function for each dimension member which does not change from one structure version into another with $w = 1$.

Mapping functions may be applied to map data between contiguous or non contiguous structure versions. Two structure versions SV_i and SV_k are contiguous if $T_{s,i} = T_{e,k} + Q$ or if $T_{s,k} = T_{e,i} + Q$.

For a split or a merge operation we need several mapping functions, e. g. , if department A splits up into A_1, A_2 and A_3 we would need three functions to map data from A to A_n and three functions to map data from A_n to A. We do not restrict the user regarding the weighting factor w. This means that the sum of all weighting factors for all functions $A \rightarrow A_n$ (split) does not have to be 1, i. e. , 100%. Vice versa not all weighting factors of the functions $A_1 \rightarrow A, ..., A_n \rightarrow A$ (merge) need to be 1.

The example given in Fig. 2 shows several structural changes in a dimension *"Divisions"*, e.g., *"Div.A"* splits up into *"Div.A_1"* and *"Div.A_2"* from SV_2 to SV_3 and that for the fact "Turnover" *"Div.A_1"* in SV_3 corresponds to 30% of the *"Div.A"* in SV_2 (see function 1). Or vice versa that the *"Div.A"* in SV_2 is equal to the sum of A_1 and A_2 in SV_3 (see functions 3 and 4). This example would result in the following mapping functions for the fact "Turnover":

1.) $MapF(SV_2, SV_3, DivA, DivA_1, \{\text{Turnover}\}, 0.3)$
2.) $MapF(SV_2, SV_3, DivA, DivA_2, \{\text{Turnover}\}, 0.7)$
3.) $MapF(SV_3, SV_2, DivA_1, DivA, \{\text{Turnover}\}, 1)$
4.) $MapF(SV_3, SV_2, DivA_2, DivA, \{\text{Turnover}\}, 1)$, and so on...

5.2 Transformation Matrices

On a conceptual level we can represent each multidimensional cube and the relationships between dimension members of different structure versions as matrices.

Let SV_i be a structure version with N dimensions. Each dimension D_N consists of a set $\mathcal{DM}_N^{L0}$ which represents all Level-0 dimension members of that dimension. We can represent this structure version as a $\mathcal{DM}_1^{L0} \times \mathcal{DM}_2^{L0} \times \ldots \times \mathcal{DM}_N^{L0}$ matrix.

Let SV_1 and SV_2 be two structure versions. We define a transformation matrix $T_{SV_1,SV_2,D_i,F}$ for each dimension D_i and each fact F. Where $T(d_i, d_j)$ is a number representing the weighting factor for mapping a fact F of dimension member d_i of structure version SV_1 to a fact of dimension member d_j of structure version SV_2.

These transformation matrices are merely another way of representing the information contained in the $MapF$ relation described above. We want to emphasize that the construction of these matrices is a conceptual view on the transformation. Any meaningful implementation will take into account that these matrices are usually sparse and will not implement the matrices in a naive way.

Example: Consider a cube C representing the structure defined through a structure version SV_1 with the dimensions $A = \{a_1, a_2, a_3\}$ and $B = \{b_1, b_2, b_3\}$ (a_i and b_j are dimension members). We represent the cell values for a specific fact in this cube as a matrix. Therefore, a value in this matrix represents a cell value in the given 2-dimensional cube.

$$C = \begin{array}{c} \\ b_1 \\ b_2 \\ b_3 \end{array} \begin{array}{c} \begin{array}{ccc} a_1 & a_2 & a_3 \end{array} \\ \begin{pmatrix} 3 & 7 & 5 \\ 10 & 8 & 6 \\ 20 & 13 & 5 \end{pmatrix} \end{array}$$

As mentioned above we need one transformation matrix for each dimension D_i to map data from structure version SV_1 into structure version SV_2. In the following example we split the dimension member a_1 into a_{11} and a_{12} and we merge b_1 and b_2 into b_{12}. The functions between SV_1 and SV_2 for a fact "*Fact*" are defined by the following operations:

- $MapF(SV_1, SV_2, a_1, a_{11}, Fact, 0.3)$
- $MapF(SV_1, SV_2, a_1, a_{12}, Fact, 0.7)$
- $MapF(SV_1, SV_2, b_1, b_{12}, Fact, 1)$
- $MapF(SV_1, SV_2, b_2, b_{12}, Fact, 1)$

To represent these functions we define two transformation matrices. T_A for dimension A, and T_B for dimension B:

$$T_A = \begin{array}{c|cccc} & a_{11} & a_{12} & a_2 & a_3 \\ \hline a_1 & 0.3 & 0.7 & 0 & 0 \\ a_2 & 0 & 0 & 1 & 0 \\ a_3 & 0 & 0 & 0 & 1 \end{array} \qquad T_B = \begin{array}{c|ccc} & b_1 & b_2 & b_3 \\ \hline b_{12} & 1 & 1 & 0 \\ b_3 & 0 & 0 & 1 \end{array}$$

5.3 Transformation of Warehouse Data

The goal of the transformation is to map the warehouse data (cube) of a certain structure version SV_1 to the structure of a different structure version SV_2. We first define a function to transform the cube in one dimension:

$f_{SV_1,SV_2,D,F}$ transforms the values of fact F of structure version SV_1 to the structure version SV_2 in the dimension D as follows:

$$f_{SV_1,SV_2,D,F}(C_{D=j}) = \sum_{j \in DM_{D,SV_1}} T_{SV_1,SV_2,D,F}(i,j) * C_{D=j} \text{ for all } i \in DM_{D,SV_2}$$

where C is a cube with the dimension members of SV_1 in dimension D and C' is the transformed cube where all values in the cube have been transformed to the members of the dimension D in the structure version SV_2 according to the transformation matrix T. $C_{D=j}$ is the (n-1) dimensional sub-cube of an n-dimensional cube associated with the member j in dimension D.

It is easy to see, that transforming a cube in dimension D_x first, and then in dimension D_y yields the same result as the transformation in the reverse sequence. The transformation of a fact F in a cube C from structure version SV_1 to structure version SV_2 is now defined as a sequence of functions successively transforming the cube in all dimensions D_i:

$$f_{SV_1,SV_2,F} = f_{SV_1,SV_2,D_1,F}(f_{SV_1,SV_2,D_2,F}(\cdots f_{SV_1,SV_2,D_n,F}(C_{SV_1})\ldots))$$

As seen from the observation above the result does not depend on the sequence of transformation used. Again, we emphasize that this is the specification of a transformation function, and the actual implementation will efficiently make use of the sparseness of the involved matrices, etc.

Example: By using the defined transformation functions we are now able to transform data from SV_1 into SV_2. The cube C and the transformation matrices T_A and T_B are given in the example in Sect. 5.2.

$$C' = f_{SV_1,SV_2,D_A,F}(f_{SV_1,SV_2,D_B,F}(C))$$

$$= \begin{array}{c|cccc} & a_{11} & a_{12} & a_2 & a_3 \\ \hline b_{12} & 3.9 & 9.1 & 15 & 11 \\ b_3 & 6 & 14 & 13 & 5 \end{array}$$

The matrix C' represents the cube with the structure defined through structure version SV_2 and the values of structure version SV_1.

6 Queries

When a user issues a query within such a system, he/she has to define a timepoint T_q. This timepoint specifies a certain base structure version where $T_s \leq T_q \leq T_e$ and $[T_s, T_e]$ defines the valid time interval of the base structure version.

This base structure version determines which structure has to be used for the analysis. In most cases this will be the current structure version. However, in some cases it will be of interest to use an "older" structure version. Suppose the structure versions given in Fig. 2 are valid for the following time periods and the chronon is a month:

Table 1. Valid time periods

Version	T_s	T_e
SV_1	Jan. 1998	Mar. 1998
SV_2	Apr. 1998	Jan. 1999
SV_3	Feb. 1999	Dec. 1999
SV_4	Jan. 2000	∞

We might assume the user chooses SV_4 as base structure version and requests data for March 2000 and March 1999 for the analysis. In this case the system needs functions to map data which is valid for the structure version SV_3 into the structure version SV_4. The same analysis however could also be made with SV_3 as base structure version. For this query the system needs functions to map data from SV_4 to SV_3.

For each query, the systems checks which structure versions are necessary to answer the query. E. g., for SV_4 as base structure version and the valid time intervals according to Tab. 1, the structure versions SV_4, SV_2 and SV_1 are necessary to answer the query "return costs for all divisions for January 1999 and January 1998". For each fact the system checks for a mapping function from SV_1 to SV_4 and from SV_2 to SV_4.

7 Conclusion

We presented a novel approach for representing changes in dimension data of multi-dimensional data warehouses, by introducing temporal extension, structure versioning and transformation functions. This representation can then be used to pose queries (analysis) against the structure valid at a given point in time and correctly admit data from other periods into the computation of the result.

This effort is necessary as changes in these data have the combined characteristics of temporal databases and schema evolution, as these dimension data serve in multi-dimensional systems as data as well as schema elements. Our approach thus overcomes the implicit orthogonality assumption underlying multidimensional data warehouses.

The transformation function we propose here can only be seen as a first step and will be elaborated in the future. The simple transformation matrices however proved themselves surprisingly powerful. We were able to represent several cases of structural changes with these data (at least approximatively). Changes which can be covered by our model comprise:

- Changes in the organizational structure of enterprises, of the regional structure of distribution systems, of product portfolios, etc.
- Changes of Units, like actually the changes from ATS to EURO.
- Changes in the way economic figures like unemployment rate, consumer price index, etc. are computed.

We also expect that our approach improves the correctness of interpretation of answers to OLAP queries and relieves the user from the need to have detailed knowledge about the change history of dimension data. In particular, our approach provides for multi-period comparisons of facts which currently requires stability in dimension data.

References

[1] M. Blaschka. FIESTA: A Framework for Schema Evolution in Multidimensional Information Systems. In *Proc. of 6th Doctoral Consortium*, Germany, 1999.

[2] M. Blaschka, C. Sapia, and G. Höfling. On Schema Evolution in Multidimensional Databases. In *Proc. of the DaWak99 Conference*, Florence, Italy, 1999.

[3] M. Böhlen. Temporal Database System Implementations. *SIGMOD*, 24(4), 1995.

[4] P. Chamoni and S. Stock. Temporal Structures in Data Warehousing. In *Data Warehousing and Knowledge Discovery (DaWaK) 1999*, p. 353–358, Italy, 1999.

[5] S. M. Clamen. Schema Evolution and Integration. In *Distributed and Parallel Databases: An International Journal*, p. 2(1):101–126.

[6] O. Etzion, S. Jajodia, and S. Sripada, editors. *Temporal Databases: Research and Practise*. LNCS 1399. Springer-Verlag, 1998.

[7] Goralwalla, Tansel, and Zsu. Experimenting with Temporal Relational Databases. *ACM*, CIKM95, 1995.

[8] Gregersen and Jensen. Temporal Entity-Relationship Models - a Survey. *TimeCenter*, 1997.

[9] C. S. Jensen and C. E. Dyreson, editors. *A consensus Glossary of Temporal Database Concepts - Feb. 1998 Version*. Springer-Verlag, 1998. In [6].

[10] C. Li and X. Wang. A Data Model for Supporting On-Line Analytical Processing. *ACM*, CIKM 96, 1996.

[11] C. Liu, S. Chang, and P. Chrysanthis. Database Schema Evolution using EVER Diagrams. In *Proc. of the Workshop on Advanced Visual Interfaces*, 1994.

[12] A. Mendelzon and A. Vaisman. Temporal Queries in OLAP. In *Proc. of the 26th VLDB Conference, Egypt*, 2000.

[13] SAP America, Inc. and SAP AG. Data Modelling with BW - ASAP for BW Accelerator. 1998. White Paper: URL: http://www.sap.com.

[14] P. Vassiliadis and T. Sellis. A Survey of Logical Models for OLAP Databases. In *SIGMOD Record 28*, 1999.

Determining the Convex Hull in Large Multidimensional Databases

Christian Böhm and Hans-Peter Kriegel

University of Munich, Oettingenstr. 67, D-80538 Munich, Germany
{boehm,kriegel}@informatik.uni-muenchen.de

Abstract. Determining the convex hull of a point set is a basic operation for many applications of pattern recognition, image processing, statistics, and data mining. Although the corresponding point sets are often large, the convex hull operation has not been considered much in a database context, and state-of-the-art algorithms do not scale well to non main-memory resident data sets. In this paper, we propose two convex hull algorithms which are based on multidimensional index structures such as R-trees. One of them traverses the index depth-first. The other algorithm assigns a priority to each active node (nodes which are not yet accessed but known to the system), which corresponds to the maximum distance of the node region to the tentative convex hull. We show both theoretically as well as experimentally that our algorithms outperform competitive techniques that do not exploit indexes.

1 Introduction

Multidimensional data sets are prevalent in many modern database applications such as multimedia [12], CAD [20], medical imaging [24], molecular biology [23], and the analysis of time sequence data [1]. In these applications complex objects are usually translated into vectors of a multidimensional space by a *feature transformation* [31]. The distance between two different *feature vectors* is a measure for the similarity of the corresponding objects. Therefore, feature vectors are often used in similarity search systems [3] where the central task is the search for objects which are most similar to a given query object. Similarity queries are transformed into neighborhood queries in the feature space, which can be efficiently supported by multidimensional index structures. Therefore, multidimensional indexes are often maintained to support modern database applications.

If the user wants to get a deeper insight into the intrinsic structure of the data stored in the database, the similarity search is not sufficient. The user will rather want to run statistical analyses or apply data mining methods such as classification [25], clustering [29], outlier detection [22], trend detection [11], or the generation of association rules [2]. Several of the corresponding algorithms use similarity queries as *database primitives*, others mainly rely on other basic operations.

The determination of the convex hull is an example of a primitive operation which is useful for many analysis methods and has successfully been applied in application domains such as pattern recognition [4], image processing [28] or stock cutting and allocation [13]. The convex hull of a point set in the plane is defined as the smallest convex polygon containing all points. Intuitively, the convex hull is obtained by spanning a rubber band around the point set. Both, the points touched by the rubber band as well as the shape of the resulting polygon are called the *convex hull* of the data set (cf. fig. 1). In 3 and higher dimensional data spaces, the convex hull is analogously defined as the minimum convex polyhedron (polytope) of the point set.

We briefly sketch a few applications of the convex hull: The convex hull is an exact and comprehensive description of the shape of a cluster of data points and can therefore be used as a postprocessing step to cluster analysis algorithms [29]. Several clustering algorithms even use the convex hull in the definition of the notion of a cluster: E.g. [7] defines a cluster to be a set of points with minimum diameter of the corresponding convex hull. The determination of the convex hull becomes thus a primitive operation for the clustering algorithm. Another application of the convex hull is the robust estimation [15]. A robust estimator (or Gastwirth-estimator) is based on the observation that points in the "inner" of a point set are generally more trustable than the extreme points. Therefore, the points of the convex hull are removed from the data set as a preprocessing step [19]. A further application of convex hull is the *isotonic regression*. Regression methods approximate point sets by functions from a given class, e.g. linear functions, such that the approximation error (least square) is minimized. In the isotonic regression, the function class are monotonic staircase functions. The isotonic regression of a point set can be found using the convex hull of the points after a transformation. A recent application of the convex hull is the Onion Technique [9] for linear optimization queries. This method is based on a theorem that a point which maximizes an arbitrary multidimensional weightening function can be found on the convex hull of the data set. The authors improve the search for such maximum points by separately indexing the points of the convex hull of the data set. Börzsönyi, Kossmann and Stocker propose the Skyline technique [5] which is a concept similar to the convex hull. The skyline of a dataset can be used to determine various poitn of a data sets which could optimize an unknown objective in the user's intentions. E.g. users of a booking system may search for hotels which are cheap and close to the beach. The skyline of such a query contains all possible results regardless how the user weights his criteria *beach* and *cost*. The skyline can be determined in a very similar way as the convex hull. The algorithms proposed in this paper can also be adapted for the skyline.

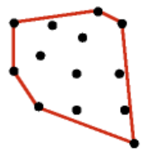


Fig. 1. Convex hull.

Due to the high practical relevance of the convex hull, a large number of algorithms for determining the convex hull in 2 and higher dimensional space has been proposed [4, 7, 8, 10, 16, 17, 21, 26, 27, 30]. Most of these algorithms, however, require the data set to be resident in main memory. In contrast, we will propose two algorithms to determine the convex hull of a point set stored in a multidimensional index. In this paper, we focus on the important case of point sets in the plane.

The remainder of this paper is organized as follows: In section 2, we introduce two algorithms determining the convex hull of a point set in a multidimensional index and state some important properties of our solutions. Section 3 evaluates our technique experimentally and section 4 concludes the paper.

2 Convex Hulls in Large Databases

In this section, we will introduce our two algorithms for the determination of the convex hull of a large point set which is stored in a multidimensional index. For our methods, we need hierarchical index structures such as the R-tree [18] or its variants. Our algorithms, however, do not exploit particular properties of R-trees such as the fact that the page regions are minimum bounding rectangles. Therefore, our algorithms are easily adaptable to index structures which use non-bounding rectangles or other shapes such as spheres or polygons. Basic operations which must be efficiently supported are the intersection with polylines, and the minimum distance to polylines and points. We will first concentrate on the distance-priority algorithm and later introduce the depth-first algorithm. In both cases, we will prove important properties of the algorithms. We will show that the distance priority algorithm yields a minimum number of page accesses. For the depth-first algorithm, we will provide worst-case bounds for the worst case time complexity.

For all algorithms, we apply the following simplification: Before starting the actual convex hull algorithm, we search the points which are extreme (maximum and minimum) in x- and y-direction (cf. figure 2). These 4 points, called *min-x-point, min-y-point, max-x-point*, and *max-y-point*, are guaranteed to be included in the convex hull. These points define 4 quadrants in which the convex hull is searched separately. In our description, we will restrict ourselves to the part of the convex hull between the min-x-point and the min-y-point. The advantage is that every line segment of the hull is oriented from upper left to lower right. Similar properties are valid for the other quadrants.

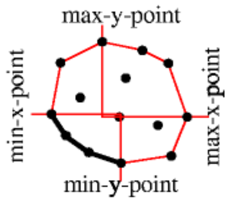


Fig. 2. Quadrant Restriction

2.1 The Distance Priority Algorithm

For developing algorithms for query processing upon multidimensional index structures it is recommendable to determine the conditions under which a page must be accessed. Figure 3 shows the situations before (upper) and after (lower) the run of a convex hull algorithm. Initially only the min-x-point and the min-y-point is known. This knowledge, however, is enough to exclude p_5 definitely from processing, because p_5 cannot contain a point on the right of the current convex hull. On the lower side of figure 3 we recognize that all points of the convex hull are located on the pages p_1 and p_2. These pages are obviously necessary for the determination of the convex hull. But it is also necessary to know the content of page p_3 to decide whether the result is correct, because p_3 could contain a point in the shaded area near by the lower left corner which belongs to the convex hull. The pages p_4 and p_5, in contrast, are topologically completely contained in the convex hull polygon and are, therefore, not able to hold any points which could be part of the convex hull. This observation leads us to the first lemma:

Lemma 1. A correct convex hull algorithm must access at least the pages which are topologically not completely contained in the convex hull polygon.

Proof. If a page which is not completely contained in the convex hull polygon, is not accessed, this page could store a point which is also not contained in the determined convex polygon. Then, the determined polygon is not the convex hull.

With our restriction to the quadrant between the min-x-point and the min-y-point it is easy to determine the pages which are not completely contained in the convex hull: The lower left corner of such regions is always left from the convex hull. If the convex hull is organized by e.g. an AVL-tree, the corresponding line segment can be found in O(log n) time.

Lemma 1 provides a stopping criterion for CH algorithms on a multidimensional index: We are done whenever all pages have been processed which are not contained in the convex hull. Our next lemma will give us the sort order in which the pages must be accessed. This lemma identifies the distance between the lower left corner of the page region and the *tentative convex hull* to be the key information. The tentative convex hull (*TCH*) is always the convex hull of all points which have already been processed. At the beginning, it is initialized with the line segment connecting the min-x-point and the min-y-point.

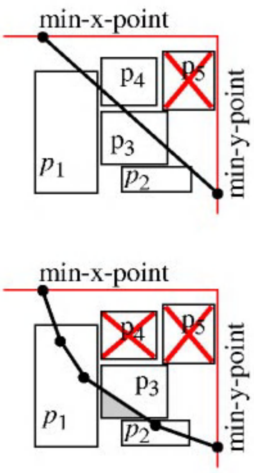
Lemma 2. If there is more than one page which is not completely contained in the TCH it is not possible that the page with the highest distance from the TCH is excluded by the convex hull.

Proof. To exclude a page p_1 from processing requires an unprocessed point v which has a higher distance from the TCH than the lower left corner of p_1. Let p_2 be the page on which the v is stored. Then, the distance between the TCH and the lower left corner of p_2 is at least as high as the distance between v and the TCH. Such a page cannot exist, because the lower left corner of p_1 has maximum distance from the TCH among all pages.

Lemma 2 can also be visualized with figure 3. Page p_1 is the farthest page from the TCH. It is not possible that any point on p_2, p_3, p_4, or p_5 extends the TCH so much that p_1 will be completely contained. We use the result of lemma 2 for our first algorithm (cf. figure 3) which manages an active page list (APL) for all pages which are not excluded by the TCH. The pages are ordered by decreasing distance between the lower left corner and the TCH. Data pages which are not excluded are basically processed as in Preparata's online algorithm [26] by determining all points which are outside the TCH. Such points are inserted into the TCH in which case it could be necessary to discard neighboring points from the TCH to remove inconvexities. The correctness of our algorithm can be guaranteed by showing that at least all points of the convex hull are passed to the Preparata's algorithm, which has been shown to be correct elsewhere [27].

Lemma 3. The algorithm *distance_priority_ch* is correct.

Proof. As the TCH is always the convex hull of a growing subset of the actual set of points, no point of the TCH can be left from the actual convex hull. Every page is processed unless it has been pruned or one of its hierarchical predecessors has been pruned. If the page has been pruned, then it must have been right from the TCH at some stage of processing, and, therefore, it must be right from the actual convex hull. If one of the predecessors has been discarded, the predecessor must be right from the actual convex hull. If a predecessor is right from the convex hull, the page must also be right from it, because the region of the page is completely contained in the region of the predecessor. Thus, each page which is not completely contained in the convex



```
ALGORITHM distance_priority_ch (root: page)
    apl := {root} ;
    minxpoint := search database for minimum x ;
    minypoint := search database for minimum y ;
    tch := {minxpoint,minypoint};
    WHILE NOT is_empty (apl)
        b := element of apl with max. distance to tch ;
        DEQUEUE b FROM apl ;
        LOAD b FROM DISK ;
        IF is_data_page (b)
            FOR EACH v ELEMENT OF b
                IF is_left (v, tch)
                    tch := tch UNION v ;
                    WHILE NOT convex (tch)
                        REMOVE next(v) FROM tch ;
        ELSE
            s := SET OF CHILD PAGES OF b ;
            apl := apl UNION s ;
        FOR ALL PAGES p ELEMENT OF apl
            IF is_right (p,tch)
                apl := apl \ p ;
```

Fig. 3. The distance priority algorithm for the CH

hull, is processed. Every point of the convex hull is contained in a page that is not completely contained in the convex hull. Therefore, every point of the convex hull plus some additional points from the data set are fed into Preparata's algorithm. Therefore, the correctness is guaranteed.

Now we will discuss the performance of our algorithm from a theoretical point of view. Our first statement is that our algorithm yields an optimum number of page accesses when assuming that the index is given.

Lemma 4. The distance priority algorithm yields a minimum number of page accesses.

Proof. According to lemma 1, all pages must be processed which are not completely contained in the convex hull. In each step of the algorithm, the page $p_{\min}$ with the highest distance between the convex hull and the lower left corner of $p_{\min}$ is processed. According to lemma 2, the convex hull cannot be extended such that $p_{\min}$ is contained in it. The algorithm accesses exactly the pages which are not completely contained in the convex hull.

Finally, we will show that our distance priority algorithm yields a worst-case complexity of $O(n \log n)$ for the CPU-time. This complexity is also the lower bound for algorithms which are not based on an index.

Lemma 5. The CPU time complexity of *distance_priority_ch* is in $O(n \log n)$.

Proof. In each step of the algorithm, one page of the index is processed. The number of index pages is linear in the number of stored points. We assume the APL to be organized in a priority queue, which requires $O(\log n)$ time for each inserted or removed element. The tentative convex hull is organized in an AVL-tree or 2-3-tree which allows fast access in $O(\log n)$ time to the y-coordinates of the stored points. Therefore, insertions and deletions can be performed in $O(\log n)$ time. The decision whether a new point is left or right from the tentative convex hull can be made in $O(\log n)$ time, because it requires the search of the points in the TCH with the next higher and the next lower y-candidates. If the point is inserted to the TCH, some neighboring points may be deleted from the TCH, which requires again $O(\log n)$ time for each deletion. During the run of the complete algorithm, at most n points can be inserted and deleted from the TCH. Taking together, both the management of the APL and the management of the TCH require O(n log n) time during the complete runtime of the algorithm.

2.2 The Depth-First Algorithm

Although the distance priority algorithm can be proven to be optimal with respect of the number of accessed pages, it does not affect the worst-case boundaries for the CPU runtime which are O($n \log n$) for our algorithm as well as for known algorithms which are not database-algorithms. There is a theoretical result that O($n \log n$) is also the lower limit for the worst-case complexity of the convex hull operation.This limit, however, is only valid for points which are not stored in an index. We may wonder whether the existence of a multidimensional index enables us to improve the worst-case complexity? This would be no contradiction to the theoretical result, because constructing an index requires O($n \log n$) time. Additionally, it is possible to extract the points in the order of ascending x- or y-coordinates from an index in linear time. The question is, whether all of the worst-case complexity is subsumed in the operation of index construction, or if there is a substantial remaining worst-case complexity. It seems intuitively clear that we cannot do better than O(n), because n is the size of the result set in the worst case. In this section, we will develop our depth-first algorithm, and we will show that this algorithm runs in O(n) time in the worst case. The general idea is to process the pages in a sequence order that corresponds to the sequence order of the points in the convex hull, starting at the min-x-point and ending at the min-y-point.

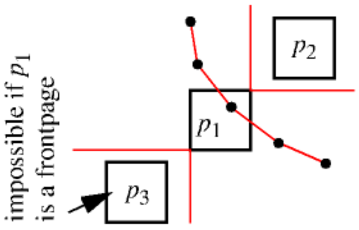


Fig. 4. Front pages and excl. pages.

Therefore, the points are inserted into and deleted from the tentative convex hull only at the end, and no points up to the end point is ever accessed. This saves the O(log n) factor in the management of the tentative convex hull (TCH). The other O(log n) factor for the management of the active page list is saved by the depth-first traversal itself. In this context, the most thrilling question is whether or not a sequence order of pages corresponding to the sequence order of the convex hull exists at all and whether or not this sequence order is compatible with a depth-first traversal through the tree. Both questions are not self-evident, because the page regions are objects yielding a spatial position *and* extension which makes it difficult to apply any ordering to them. E.g. points could be uniquely ordered by x- or y-coordinate or by their angle and distance with respect to some appropriate reference point. Such kinds of ordering have been successfully applied for traditional convex hull algorithms. For extended spatial objects, however, the x- and y-projections of different objects overlap, as well as the corresponding angle does. We will develop in the first two lemmata a new appropriate ordering for page regions and show that it is compatible with the sequence order of the convex hull. For these two lemmas, we assume overlap-free indexes. Note that some index structures allow overlap among sibling page regions to improve the adaptation to the data distribution in the presence of heavy update to the database. For unlimited overlap (i.e. arbitrarily bad indexes), obviously our method cannot guarantee a general O(n) time complexity in the worst case. Our lemma gives us a statement to exclude some pages from processing even if the TCH in this area is not yet known. We still restrict our convex hull to the quadrant between the min-x-point and the min-y-point. For other quadrants, analogous properties hold.

Lemma 6. If a page p_2 is completely in the sector above and right from another page p_1, this page cannot contain any point of the convex hull.

Proof. The convex hull cannot be left or below from any point which is stored in the page p_1. Therefore, p_1 is either completely contained in the convex hull or it is intersected by it. In our quadrant, all line segments of the convex hull yield a negative skew. Therefore, the convex hull cannot intersect p_2.

For an illustration of lemma 6, cf. the pages p_1 and p_2 in figure 4. In any set of non-overlapping pages, some pages exclude other pages from processing according to lemma 6. We call those pages which are not excluded the *frontpages* of the set. We know for each frontpage p_1 that no other frontpage is completely in the sector above and right (such as p_2) or completely in the sector below and left (such as p_3) of the page. In the first case, p_2 is excluded by p_1. In the second case, p_3 would exclude p_1 such that p_1 cannot be a frontpage.

Definition 1. (Frontpage ordering):
A frontpage p_2 is greater than a frontpage p_1 (or equal) with respect to frontpage ordering ($p_2 \geq_{\text{fpo}} p_1$) if at least one of the following conditions hold:

- p_2 is completely right from p_1
- p_2 is completely below from p_1
- p_1 and p_2 are identical

Lemma 7. The relation "$p_2 \geq_{\text{fpo}} p_1$" for non-overlapping frontpages p_1 and p_2 is a total ordering.

We briefly sketch the idea of the proof which is technically complex but not very interesting. The reflexivity is obvious. The antisymmetry is not given for pages which yield overlap or which don not fulfill the frontpage property: If, for instance, p_1 is both, above and right from p_1, we have $p_2 \geq_{\text{fpo}} p_1$ and $p_1 \geq_{\text{fpo}} p_2$ which would violate the antisymmetry. For overlap-free frontpages, in contrast, it follows that p_1 is either above, or right from p_2 but not both (in this case, p_2 would exclude p_1). Similarly, p_2 is either below or left from p_1 but not both. In all cases, we cannot have $p_2 \geq_{\text{fpo}} p_1$ and $p_1 \geq_{\text{fpo}} p_2$ for two different pages p_1 and p_2. For the transitivity, a similarly complex argumentation with many case distinctions is necessary, but viable.

Next, we will prove the most important lemma for our depth-first algorithm which claims that the frontpage ordering is compatible with the ordering of points in the convex hull. The consequence is that if we process the pages by ascending frontpage ordering, we get the points of the convex hull in the right order.

Lemma 8. For two points $v_1=(x_1,y_1)$ and $v_2=(x_2,y_2)$ of the convex hull which are stored on different pages p_1 and p_2, the following statements are equivalent:

(1) $p_2 \geq_{\text{fpo}} p_1$

(2) $x_2 \geq x_1$

(3) $y_2 \leq y_1$

Proof. (2) $\Leftrightarrow$ (3): The equivalence of statement (2) and (3) is an immediate consequence of the properties of the convex hull and our restriction to the quadrant between the min-x-point and the min-y-point.

(1) $\Rightarrow$ (2): Case (a): p_2 is completely right from p_1. In this case, all points stored on p_2 have higher x-coordinates than any point stored in p_1, and therefore, $x_2 \geq x_1$.

Case (b): p_2 is completely below p_1. In this case, all points stored on p_2 have lower y-coordinates than any point stored in p_1, and therefore, $y_2 \leq y_1$. Due to the equivalence of statement (2) and (3), we know that $x_2 \geq x_1$.

(2) $\Rightarrow$ (1): We know that p_2 stores a point with a higher x-coordinate than a point that is stored in p_1. Therefore, p_2 cannot be completely left from p_1. Due to "(2)$\Leftrightarrow$(3)" we further know that p_2 stores a point with a lower y-coordinate than a point that is stored in p_1. So p_2 cannot be completely above p_1. In order to be overlap-free, p_2 must either be completely right from p_1 or completely below. In both cases it follows that $p_2 \geq_{\text{fpo}} p_1$.

Last before we are ready to present our depth-first algorithm, we state that the frontpage ordering is compatible with the depth-first traversal of the tree:

Lemma 9. If $p_2 \geq_{\text{fpo}} p_1$ (but not $p_2 = p_1$) then also $c_{2,i} \geq_{\text{fpo}} c_{1,i}$ for all child pages $c_{1,i}$ of p_1 and $c_{2,i}$ of p_2 which are frontpages.

Proof. Follows from definition 1 and as child pages are contained in the region of their parent.

From our set of lemmata it follows that we can traverse the tree in a depth-first fashion, in each node calling those child nodes which are front pages, ordered by the frontpage order. If we do so, we find the points of the convex hull in the right order. This property is exploited in our algorithm depth_first_ch (cf. figure 6). Here, the part of the algorithm which corresponds to the conventional hull determination (IF is_data_page...) corresponds to Graham's scan. The part of the algorithm for the directory pages first determines those child pages which are frontpages. Then these pages are ordered according to the frontpage ordering relation and accessed unless they can be pruned.

This section is concluded by two lemmata stating the correctness and the worst-case runtime of the algorithm which is proven to be in O(n).

Lemma 10. The algorithm depth_first_ch is correct.

Proof. Analogously to lemma 3, we can show that every point of the convex hull is passed to the conventional convex hull algorithm. The variant used in our algorithm requires the points of the convex hull to be passed in the appropriate order, i.e. with ascending x-coordinates and descending y-coordinates. This is follows from lemmata 8 and 9.

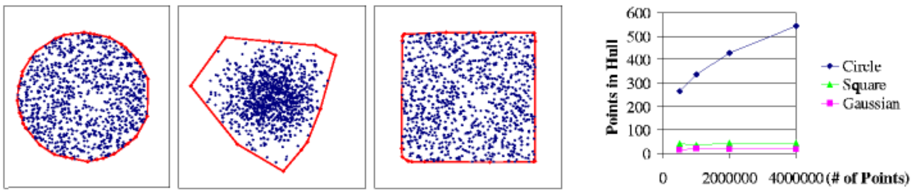


Fig. 5. Convex hulls and their characteristics: circle, gaussian and square

```
ALGORITHM depth_first_ch (b: page)
  LOAD b FROM DISK ;
  IF is_data_page (b)
    V := set of points stored on b ;
    ORDER V BY x-coordinate ASCENDING ;
    FOR EACH v ELEMENT OF V
      l := linesegment connecting the last
          point of the tch with minypoint ;
      IF is_left (v, l)
        m := linesegment connecting v with
            the point before last point of tch
        WHILE is_right (last point of tch, m)
          DELETE last point FROM tch ;
        APPEND v TO tch ;
  ELSE (* directory page *)
    P := set of child pages of b ;
    (* restrict P to the frontpages: *)
    FOR EACH p ELEMENT OF P
      FOR EACH q ELEMENT OF P
        IF excludes (p,q)
          REMOVE q FROM P ;
    ORDER P BY "$\geq_{\text{fpo}}$" ASCENDING ;
    FOR EACH p ELEMENT OF P
      l := linesegment connecting the last
          point of the tch with minypoint ;
      IF NOT is_right (p,l)
        depth_first_ch (p) ;
```

Fig. 6. The depth first algorithm for the CH

Lemma 11. The algorithm depth_first_ch runs in O(n) time in the worst case, if the data set is stored in an overlap-free multidimensional index.
Proof. In the worst case, each page of the index is accessed once. The number of pages is linear in n. For each page, only a constant number of operations is raised. The only exception is the deletion of the last point of the TCH (in constant time) which is repeated until the convexity of the TCH is maintained. The total number of deletions during the run of the algorithm, however, is also restricted by n.

3 Experimental Evaluation

To demonstrate the practical relevance of our technique and the superiority over competitive approaches we implemented our distance priority algorithm, two variants of depth-first algorithms, and two related approaches, the scalable variants of Graham's scan as well as Preparata's online algorithm. For efficiency all variants have been coded in C and tested on HP C160 workstations under HP-UX 10.12. The data set was stored on a disk with 5 ms seek time, 5 ms rotational delay, and a transfer rate of 4 MByte per second.

All implementations of our new index-based algorithms operate on bottom-up constructed X-trees [6]. We implemented two variants of depth-first algorithms. The first variant processes the child nodes of the current node ordered by the front-page ordering as described in section 2.2 (in the following called *depth-first with FPO*). In the second variant the child nodes are ordered by their maximum distance to the TCH (*depth-first with priority*). In contrast to the distance priority algorithm, this algorithm traverses the index depth-first, i.e. every branch is completed before the next branch can be started. Therefore, it is not necessary to manage a priority-queue. To be scalable to non-memory databases, Graham's scan [16] was implemented using the mergesort algorithm

For our experiments we used 3 data sets with 4,000,000 data points with different distribution characteristics. The first dataset contains points which are uniformly distributed in a unit circle. The points of the second data set are normally distributed (*Gaussian data set*). The third set contains uniformly distributed points in the unit square. All data sets and their convex hulls (for 1,000 points) are depicted in figure 5.

The characteristics of the convex hull are depicted on the right side of figure 5. The number of points in the convex hull of the circle data set is fast yet sublinearly increasing with increasing data set size. In contrast the number of points in the convex hull of the other data sets is not much influenced by the database size.

In all subsequent experiments, the number of data points varied between 500,000 and 4,000,000. The block size of the index and data files was consistently set to 2 KBytes. Graham's scan is the only algorithm which needs a page cache for processing (for the sorting step). We constantly allowed 800 database pages in cache which is between 5% and 40% of the database.

Figure 7 depicts the results of the experiments on the circle data set. In the left figure, the number of page accesses of the different algorithms is compared. On the circle data set, the distance priority algorithm and the two depth-first algorithms required exactly the same number of page accesses (up to 530). In contrast, Graham's scan needs 48,000 page accesses for sorting of the data set. The online algorithm by Preparata is with 16,000 block accesses also clearly outperformed. The diagram in the middle shows the CPU times. Here the depth-first variant with frontpage ordering yields the best performance with 1.1 seconds for 4,000,000 data points. As the circle data set has many points in the convex hull, the reduced effort for the management of the TCH (which is a consequence of frontpage ordering because points are only inserted or deleted at the end of the TCH) results in substantial performance gains. The depth-first variant needed 4.4 seconds of CPU time or four times as much as depth-first with FPO. Slightly worse (5.1 sec) due to the management of the priority queue was the distance priority algorithm. The variants without index, however, needed with 43 (Preparata) and 110 (Graham) seconds, respectively, a factor of 39 (100) times more CPU power than our best performing algorithm. The right diagram shows the total time subsuming CPU time and I/O time. As all algorithms are I/O bound the differences between the index-based algorithms are alleviated. For 4,000,000 data points, the FPO algorithm needed 6.2 seconds, the other depth-first algorithm needed 9.6 seconds and the distance priority algorithm needed 10.36 seconds. With 200 seconds, the online algorithm was outperformed with factor 32. The improvement factor over Graham's scan (580 seconds) was even 94.

The characteristic of the square data set which is depicted in figure 8 is quite different from the characteristic of the circle data set. The number of points in the convex hull is with 43 very small. Therefore, the management of the convex hull does not play an important role and the advantage of frontpage ordering over our other index-based approaches disappears. For this set, the distance priority algorithm clearly yields the best number (120) of page accesses. With 650 page accesses, the

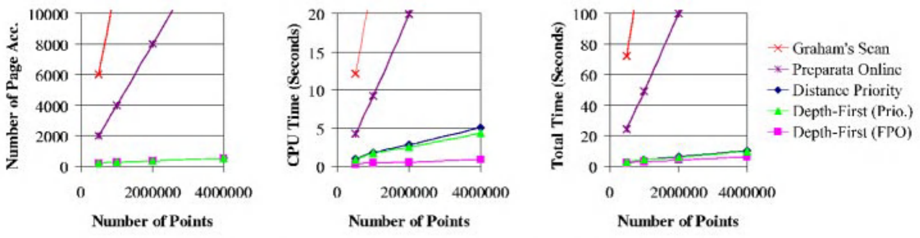


Fig. 7. Experimental results of the circle data set

depth-first algorithm is clearly worse than the distance priority algorithm, but still very good in comparison with the algorithms without index. Ordering child pages by priority rather than by FPO which is a kind of compromise between the other two approaches yields a negligible advantage (630 page accesses). As the TCH management does not play an important role and since processing of each page needs CPU power, the distance-priority algorithm yields also the best CPU performance. With respect to the overall time, the distance priority algorithm outperforms the depth-first variant with priority ordering by a factor 5.1, the depth-first algorithm with FPO by 5.5, the online algorithm by 130, and Graham's scan by the factor 420.

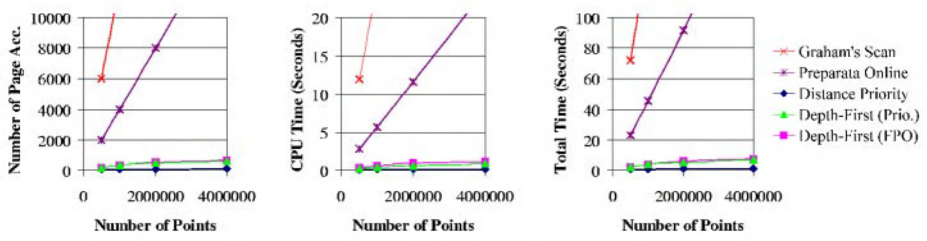


Fig. 8. Experimental results of the square data set

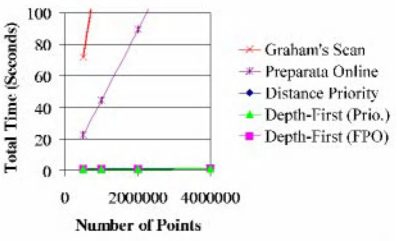


Fig. 9. Experiments on Gaussian data set

These results are also confirmed by the gaussian data set . The corresponding experiment (total time only) is shown in figure 9. With 1.3 sec. (distance priority) and 1.2 sec (both depth-first algorithms), respectively, all index based approaches required about the same time. In contrast, Preparata's online algorithm needed 180 seconds, i.e. more than 150 times slower than our approaches. With 590 seconds, Graham's scan is even outperformed by factor 510. Like in all our experiments, the improvement factors of our new techniques over competitive techniques was increasing with increasing database sizes. For instance, the improvement factor over the online algorithm (Gaussian data set) starts with 25 for 500,000 data points. For 1 million points, the factor increases to 52, then 89 (2 million points), and finally 150. The analogous sequence for the improvement over Graham's scan is (80, 170, 291, 510).

4 Conclusions

In this paper, we have proposed to use multidimensional index structures for the determination of the convex hull of a point database. Indexes can be either traversed depth-first or the access of the next node is controlled by the priority of the node which corresponds to the maximum distance between the node and the currently available part of the convex hull. The analytical evaluation of our technique shows that the distance priority algorithm is optimal with respect to the number of disk accesses. The depth-first algorithm, in contrast, has a better (i.e. linear) worst-case complexity with respect to CPU time. Our experimental evaluation demonstrates the superiority over approaches storing the point set in flat files. The database implementation of the most well-known convex hull algorithm, Graham's scan, is outperformed by factors up to 510.

Acknowledgments. We are grateful to Donald Kossmann for fruitful discussions.

References

1. Agrawal R., Faloutsos C., Swami A.: *Efficient similarity search in sequence databases*, Int. Conf. on Found. of Data Organization and Algorithms, 1993.
2. Agrawal R., Imielinski T., Swami A.: *Mining Association Rules between Sets of Items in Large Databases*, ACM SIGMOD Int. Conf. on Management of Data, 1993.
3. Ankerst M., Kriegel H.-P., Seidl T.: *A Multi-Step Approach for Shape Similarity Search in Image Databases*, IEEE Transactions on Knowledge and Data Engineering (TKDE), Vol. 10, No. 6, 1998.
4. Akl S. G., Toussaint G. T.: *Efficient Convex Hull Algorithms for Pattern Recognition Applications*, Int. Joint Conf. on Pattern Recognition, 1978.
5. Börzsönyi S., Kossmann D., Stocker K.: *The Skyline Operator*, Int. Conf on Data Engineering, 2000.
6. Berchtold S., Böhm C., Kriegel H.-P.: *Improving the Query Performance of High-Dimensional Index Structures Using Bulk-Load Operations*, Int. Conf. on Extending Database Technology, 1998.
7. Brown K. Q.: *Geometric Transformations for Fast Geometric Algorithms*, Ph.D. thesis, Dept. of Computer Science, Carnegie Mellon Univ., Dec. 1979a.
8. Bykat A.: *Convex Hull of a Finite Set of Points in Two Dimensions*, Info. Proc. Lett., No. 7, 1978.
9. Chang Y.-C, Bergman L. D., Castelli V., Li C.-S., Lo M.-L., Smith J. R.: *The Onion Technique: Indexing for Linear Optimization Queries*, ACM SIGMOD Int. Conf. on Management of Data, 2000.
10. Eddy W.: *A New Convex Hull Algorithm for Planar Sets*, ACM Trans. Math. Software 3 (4), 1977.
11. Ester M., Frommelt A., Kriegel H.-P., Sander J.: *Algorithms for Characterization and Trend Detection in Spatial Databases*, Int. Conf. on Knowledge Discovery and Data Mining (KDD), 1998.
12. Faloutsos C., Barber R., Flickner M., Hafner J., Niblack W., Petkovic D., Equitz W.: *Efficient and Effective Querying by Image Content,* Journal of Intelligent Information Systems, Vol. 3, 1994.
13. Freeman H., Shapira R.: *Determining the Minimum-Area Encasing Rectangle for an Arbitrary Closed Curve*, Comm. ACM, Vol 18, No. 7, 1975.
14. Gaede V., Günther O.: *Multidimensional Access Methods*, ACM Computing Surveys, 30 (2), 1998.
15. Gastwirth J.: *On Robust Procedures*, Journal Amer. Stat. Ass., Vol 65, 1966.
16. Graham R. L.: *An Efficient Algorithm for Determining the Convex Hull of a Finite Planar Set*, Info. Proc. Lett., Vol. 1, 1972.
17. Green P. J., Silverman B. W.: *Constructing the Convex Hull of a Set of Points in the Plane*, Computer Journal, Vol. 22, 1979.
18. Guttman A.: *R-trees: A Dynamic Index Structure for Spatial Searching*, Proc. ACM SIGMOD Int. Conf. on Management of Data, 1984.
19. Huber P. J.: *Robust Statistics: A Review*, Ann. Math. Stat., Vol. 43, No. 3, 1972.

20. Jagadish H. V.: *A Retrieval Technique for Similar Shapes*, ACM SIGMOD Int. Conf. Manag. Data, 1991.
21. Jarvis R. A.: *On the Identification of the Convex Hull of a Finite Set of Points,* Info. Proc. Lett., Vol. 2, 1973.
22. Knorr E. M., Ng R. T.: *Algorithms for Mining Distance-Based Outliers in Large Datasets*, Int. Conf. on Very Large Data Bases (VLDB), 1998.
23. Kriegel H.-P., Seidl T.: *Approximation-Based Similarity Search for 3-D Surface Segments*, GeoInformatica Int. Journal, Vol. 2, No. 2, 1998.
24. Korn F., Sidiropoulos N., Faloutsos C., Siegel E., Protopapas Z.: *Fast Nearest Neighbor Search in Medical Image Databases*, Int. Conf. on Very Large Data Bases (VLDB), 1996.
25. Mitchell T. M.: *Machine Learning*, McCraw-Hill, 1997.
26. Preparata F. P.: *An Optimal Real Time Algorithm for Planar Convex Hulls*, Comm. ACM 22, 1979.
27. Preparata F. P., Shamos M. I.: *Computational Geometry*, Springer New York, 1985.
28. Rosenfeld A.: *Picture Processing by Computers*, Academic Press, New York, 1969.
29. Sander J., Ester M., Kriegel H.-P., Xu X.: *Density-Based Clustering in Spatial Databases: The Algorithm GDBSCAN and its Applications*, Data Mining and Knowledge Discovery, Vol. 2, No. 2, 1998.
30. Shamos M. I.: *Computational Geometry*, Ph.D. Thesis, Dept. of CS, Yale University, 1978.
31. Seidl T., Kriegel H.-P.: *Efficient User-Adaptable Similarity Search in Large Multimedia Databases*, Int. Conf. on Very Large Data Bases (VLDB), 1997.

Shared Index Scans for Data Warehouses

Yannis Kotidis[1]*, Yannis Sismanis[2], and Nick Roussopoulos[2]

[1] AT&T Labs–Research, 180 Park Ave, P.O. Box 971 Florham Park, NJ 07932 USA
`kotidis@research.att.com`
[2] University of Maryland, College Park
`{isis,nick}@cs.umd.edu`

Abstract. In this paper we propose a new "transcurrent execution model" (TEM) for concurrent user queries against tree indexes. Our model exploits intra-parallelism of the index scan and dynamically decomposes each query into a set of disjoint "query patches". TEM integrates the ideas of prefetching and shared scans in a new framework, suitable for dynamic multi-user environments. It supports time constraints in the scheduling of these patches and introduces the notion of *data flow* for achieving a steady progress of all queries. Our experiments demonstrate that the transcurrent query execution results in high locality of I/O which in turn translates to performance benefits in terms of query execution time, buffer hit ratio and disk throughput. These benefits increase as the workload in the warehouse increases and offer a scalable solution to the I/O problem of data warehouses.

1 Introduction

Tree-based indexes like B-trees, B^+-trees, bitmap indexes [16,2] and variations of R-trees [19,12] are popular in data warehousing environments for storing and/or indexing massive datasets. In a multiuser environment accessing these indices has the potential of becoming a significant performance bottleneck. This is because in an unsynchronized execution model, concurrent queries are "competing" for the shared system resources like memory buffers and disk bandwidth while accessing the trees. Scheduling of disk requests and prefetching are well-studied techniques [8,23,1] exploited by all modern disk controllers to maximize the performance of the disk sub-system. However, optimizing the execution of the I/O at the physical disk level does not always realize the potential performance benefits. Even-though most commercial systems perform asynchronous I/O, from the buffer's perspective the interaction with a query is a synchronous one: the query thread asks for a page, waits till the buffer manager satisfies the request and then resumes execution. This leaves the buffer manager with limited opportunities for maximizing performance. In most cases, overlapping I/O among multiple queries is only exploited if it occurs within a small time-space window.

* Work performed while the author was with the Department of Computer Science, University of Maryland, College Park

Recently, data warehousing products introduced the notion of *shared circular scans* (e.g. RedBrick). The idea is for a new scan to join (merge) with an existing scan on the index/table that currently feeds a running query. Obviously the latter scan will have to access the beginning of the index later. Microsoft's SQL server supports "merry-go-round" scans by beginning each index at the current position, however there is no explicit synchronization among the queries. In this paper we capitalize on, extend and formalize the idea of shared index scans. We propose a new "transcurrent execution model" (TEM) for concurrent user queries against tree indices, which is based on the notion of detached non-blocking query patches. This is achieved by immediately processing index pages that are in memory and detaching execution of disk-resident parts of the query that we call "patches". TEM allows uninterrupted query processing while waiting for I/O. Collaboration among multiple queries is accomplished by synchronizing the detached patches and exploiting overlapping I/O among them. We use a circular scan algorithm that dynamically merges detached requests on adjacent areas of the tree. By doing the synchronization *before* the buffer manager, we manage to achieve a near-optimal buffer hit ratio and thus, minimum interaction with the disk at the first time. Compared against shared circular scans, TEM is far more dynamic, since merging is achieved for any type of concurrent I/O, not just for sequential scans of the index.

We further exploit prefetching strategies, by grouping multiple accesses in a single *composite request* (analogous to multipage I/Os in [5]) that reduces communication overhead between the query threads and the buffer manager and permits advanced synchronization among them. An important difference is that our composite requests consist of pages that will actually be requested by the query. On the contrary, typical prefetching techniques retrieve pages that have high-probability of being accessed in the future, but might be proven irrelevant to the query. Another contribution of the TEM framework is that we address the issue of fairness in the execution of the detached patches and introduce the notion of *data flow* to achieve a steady flow of data pages to all query threads. This allows efficient execution of complex query plans that pipeline records retrieved by the index scans.

The rest of the paper is organized as follows: section 2 discusses the motivation behind our architecture. In section 3 we provide a detailed description of the TEM and discuss related implementation and performance issues. In section 4 we define data flow and show how to support time constraints in the scheduling of the detached query patches. Finally, section 5 contains the experiments and in section 6 we draw the conclusions.

2 Motivation

Most commercial data warehouses maintain a pool of session-threads that are being allocated to serve incoming user queries. In a data warehouse environment, the I/O is read-only and the query threads generate concurrent read page requests.

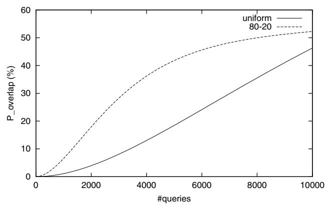


Fig. 1. Probability of overlapping I/O for uniform and 80-20 accesses

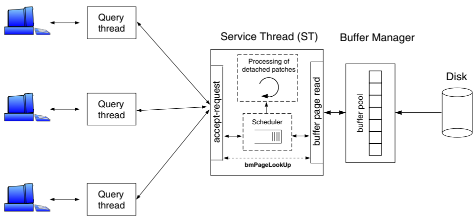


Fig. 2. System overview

In the database literature there is an abundance of research on buffer allocation and replacement strategies (e.g. [13,4,15]). For tree index structures, a Domain Separation Algorithm [17] introduced multiple LRU buffer pools, one for each level of the tree. Similar algorithms are discussed in [21,7,14]. However in [6] Chou and DeWitt point out that for indices with large fan-out the root is perhaps the only page worth keeping in memory. In data warehouses, indices are typically created and refreshed through bulk operations. As a result the trees tend to be rather packed and shallow and the potential improvements from a domain separation algorithm are limited.

In a concurrent multi-user environment there is limited potential for improving the buffering of the leaf-level pages. Given $n > 2$ concurrent queries on an 100MB index (6,400 16KB pages) and uniform distribution of accesses, the probability that the same data page is requested by 2 or more queries at any given time is:

$$p_{overlap}(n) = 1 - p(1/6400, 0, n) - p(1/6400, 1, n) \qquad (1)$$

where:

$$p(a, k, n) = \frac{n!}{k! * (n-k)!} * a^k * (1-a)^{n-k} \qquad (2)$$

is the standard binomial distribution. In Figure 1 we plot $p_{overlap}$ as the number of concurrent queries increases from 1 up to 10,000. We also plot the same probability for a more realistic 80-20 access pattern, where 20% of the pages receive 80% of the requests. These graphs show that for reasonable numbers of concurrent users, these probabilities are practically zero.

For most cases, where only a small part of the index fits in memory, overlapping requests have to occur within a relatively short time-window before the replacement strategy used flushes "idle" pages to disk. To overcome this limitation, we propose a non-blocking mechanism, in which requests for pages that are not in memory are detached from the current execution while the query thread advances its scan. We call this model of execution *transcurrent* because it detaches the processing of the requested I/O and delegates it as a *query patch* to an asynchronous *service thread*. This creates the illusion of a higher number of concurrent queries and results in increased chances of getting overlapping I/O.

3 Transcurrent Execution Model (TEM)

In this section we propose an architecture for the TEM. Our goal throughout the design was to require the least amount of modifications for integration into existing systems. The architecture introduces a new "Service Thread" (ST) for each active index, i.e. an index that is accessed by a query. This thread is interjected between the buffer manager and the query threads as shown in Figure 2.

The ST accepts all read-page requests for its corresponding index. Requests for pages that are in memory are immediately passed to the buffer manager and ST acts as a transparent gateway between the query thread and the buffer manager, as seen in the Figure.[1] For pages that are not in the buffer pool their requests are queued within the ST. The query thread that issued the request can either block until the page is actually read, or in some cases advance the index scan. In the later case the processing of the page is detached and performed internally, as a query patch, by ST when the page is actually fetched from disk.

Our goal is to optimize the execution order of the requests that are queued in ST in order to achieve higher buffer hit ratio. Assuming that the buffer manager is using a variant of the LRU replacement policy, an optimal execution strategy for ST would be to put requests on the same page together, so that subsequent reads would be serviced immediately from the pool. For that, we use a circular scan algorithm [8,23] that is frequently used for scheduling I/O at the disk controller level. By using the same type of algorithm for our synchronization step, we not only achieve our goal, which is to maximize hit ratio, but we also generate an I/O stream that is bread-and-butter for a modern disk controller.

3.1 Index Scans for TEM

In an traditional index scan, each page request will block the query thread until the page is brought in the buffers. In the TEM if the page is not in the buffer pool, then the request is queued and will be executed at a later time according to the scheduling algorithm described in the next subsection. We then have the option to block the query thread, until the page is actually fetched from the disk or let it scan the rest of the tree.

For indexes that are created using bulk-load operations in a data warehouse, non-leaf pages occupy a very small fraction of the overall index space. Assuming relatively frequent queries on the view, most of these pages will be buffered in memory. As a result there is no evidence in getting any improvement by advancing the search for non-leaf pages, since in most cases the page will be available in the buffers anyway and the scan will not be blocked. Therefore, we do not consider detaching execution of non-leaf page requests and the query thread is blocked whenever such a request is not immediately satisfied from the buffer pool. On the contrary, for the leaf (data) pages, because of the asynchronous execution,

[1] We assume that the manager provides a `bmPageLookUp`(*pageId*) function for determining whether the requested page is in the buffers.

the probability that the page is in the pool at the exact time that the request is made is very small, see Figure 1. Therefore for leaf pages it is faster to detach their requests without checking the buffers with the `bmPageLookUp`(*pageId*) function, otherwise a lot of thread congestion is happening.

3.2 Synchronization of Detached Query Patches

The ST maintains the current position on the file of the last satisfied disk I/O. Query threads continuously generate new requests, either as a result of a newly satisfied page request, or because of the non-blocking execution model that allows the search algorithm to advance, even if some page-reads are pending. Incoming page requests are split into two distinct sets. The first called "left" contains requests for pages before the last satisfied page of the index and the set called "right" (assuming a file scan from left to right) contains requests for pages infront of the last page accessed. These are actually multi-sets since we allow duplicates, i.e multiple requests for the same page by different threads. The next request to satisfy is the nearest request to the current position from the `right` set that is realized as a heap. When the `right` set gets empty, the current position is initialized to the smallest request in the `left` set and the `left` set becomes the `right` set.

An extension that we have implemented but not include in this paper is to permit pages in the `right` set, even if they are within some small threshold on the left of the current position. The intuition is that these pages are likely to be in the cache of the disk controller. This allows better synchronization of queries that are slightly "misaligned". We also experimented with an elevator algorithm that switches direction when reaching the end/start of the file. To our surprise this algorithm was much slower than the circular scan algorithm that we described, probably due to conflicts with the scheduling implemented at the hardware of our disk controller. We plan to investigate this matter on different hardware platforms.

3.3 Composite Requests for Increased Overlapping I/O

An opportunity for optimization arrives when processing nodes just above the leaves of the tree. When such a node is processed we group multiple requests for leaf pages into a single *composite* multi-page request. Notice that we do not want to apply the same technique when accessing intermediate nodes higher in the tree structure, otherwise the search is reduced to a Breadth First Search algorithm that requires memory proportional to the width of the index.

Composite requests are more efficient because of the lower overhead required for the communication between the query-thread and the ST. However, an even more important side-effect of using composite requests it that the ST is given more information at every interaction with the query threads and is able to provide better synchronization. For instance, 10 concurrent single-page requests provide at any given point 10 “hints” to the scheduler (ST) on what the future I/O will be. In comparison composite requests of 100 pages each,[2] provide a hundred times more information at any given point. Looking back at Figure 1 this generates the illusion of having $10 * 100 = 1000$ concurrent queries for which now the probability of overlap is $\frac{p_{overlap}(1000)}{p_{overlap}(10)} = \frac{1.0998e{-2}}{1.1e{-6}} = 9998$ times (i.e four orders of magnitude) higher! Furthermore, due to the non-blocking execution of the index scan, the query threads are constantly feeding the ST with more information, resulting in even higher gains. In this sense, even-though transcurrent query execution and dynamic synchronization can be seen as two orthogonal optimizations they are very naturally combined with each other.

3.4 Scheduling vs. Cache Management

In our initial designs we thought of exploiting the large number of pending requested queued in ST for better cache management in a way similar to [22]. We tested an implementation of a modified LRU policy that avoids replacing pages that have pending requests on the right set of ST. Even though this resulted in a slightly higher buffer hit ratio than plain TEM, the gains did not show up in query execution times because of the per-request overhead of synchronizing ST’s structures with the buffer manager. Our current implementation is cleaner, easier to integrate with existing systems and reorders the requests in a way that is ideal for LRU as shown in Figure 5.

4 Flow Control Extensions

Deferred requests are used in TEM as a mean to identify overlap among concurrent queries. A potential drawback is that a request that is diverted to the `left` set (see section 3.2) can be delayed while incoming requests keep pushing the request flow to the `right` set. Starvation is not possible, because the current file position is monotonically increasing up to the point that the last page of the index is read or the `right` set is exhausted, or both. However, for queries whose output is consumed by another thread, like a sub-query in a pipelined execution model; a delayed delivery of data will block the consuming thread.

In TEM, the ST executes detached leaf page requests while the query thread QT processes requests for non-leaf pages. Assuming that QT processes P_{QT} non-leaf pages per second and the ST processes P_{ST} leaf pages per second the aggregate processing for the query is: $P_{overall} = P_{QT} + P_{ST}$. Since output is only produced when scanning leaf pages, from the user point of view the effective progress of his query q that we denote as *data flow* (DF) is:

$$DF(q) = P_{ST} \tag{3}$$

Intuitively the larger this number is, the more bursty the output of the query gets. In the presence of many concurrent queries, a steady data flow for all of

[2] the fan out of a 2-dim R-tree with 16KB page size is 819. We assume that one in 8 leaves under a visited level-1 page are relevant to the query

them can be achieved by bounding the *idle time* t_{idle} of their leaf page requests. This idle time is defined as the period between two consecutive satisfied leaf page requests. The ST maintains a time-stamp information for each query that is running in the system and uses the index. This time-stamp is updated every time a leaf-page request is satisfied for the query. The administrator defines a "hint" for the maximum time W that a detached leaf-request is allowed to be delayed. Our implementation uses a *Flow Control Thread* that periodically checks for *expired* requests. Assuming that this thread awakes every T time-units and checks all time-stamps, then a query's output might block, waiting for a data page for a period of $t_{idle} = W + T$ and therefore the minimum data flow for the query will be:

$$DF_{low} = \frac{1}{W+T}\, pages/sec \tag{4}$$

Leaf page requests that have expired are inserted in a priority queue that uses the delay information for sorting them. As long as the ST finds expired requests in this queue, it processes them before those in the `right` set. The number of page requests in this queue is bounded by the number of concurrent queries in the system, as we only need at most one expired request per query to lower-bound the data flow. This prevents the data flow mechanism to become too intrusive if very small values for W and T are chosen.

5 Experiments

The experiments that we describe in this section use an implementation of TEM on top of the ADMS [18] database management system. We have used the TPC-D benchmark [9] for setting up a demonstration database. TPC-D models a business data warehouse where the business is buying `products` from a `supplier` and selling them to a `customer`. The measure attribute is the `quantity` of `products` that are involved in a transaction. We concentrate on an aggregate view for this dataset that aggregates the `quantity` measure on these three dimensions.

This view was materialized using a 3-dimensional R-tree stored in a raw disk device. We did not use regular Unix files in order to disable OS buffering. The total number of records in the view was 11,997,772 and the overall size of the R-tree was 183.8MB. The number of distinct values per attribute was 400,000, 300,000 and 20,000 for `product`, `customer` and `supplier` respectively. All experiments were ran in a single CPU SUN Ultra-60 workstation.

5.1 Comparison of TEM against an Unsynchronized Execution

For the first experiment, we executed between 10 and 200 concurrent random range queries, each running in a different thread.[3] We used three different configurations. The first, which is denoted as "CEM" in the graphs, refers to the "conventional" execution model, where all queries are unsynchronized. The second

[3] Due to space limitation we omit similar results for skewed workload

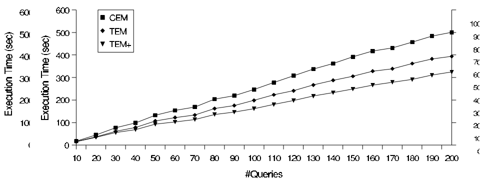


Fig. 3. Total execution time for 10-200 concurrent queries

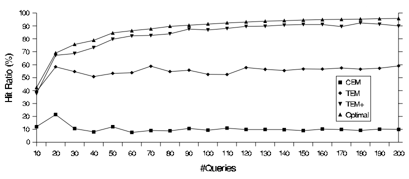


Fig. 4. Buffer hits

configuration used the transcurrent execution model with single page requests while the last one used composite requests as described in section 3.3. These configurations are denoted as TEM and TEM+ respectively. The ST maintains a pre-allocated pool of request objects that are used in the `left` and `right` sets. For the TEM/TEM+ configurations we set the request pool size to be 1MB and the buffer pool size of ADMS to 15MB. Since CEM does not use the ST, we gave the extra 1MB worth of memory to the buffer manager and set its pool size to be 16MB.

Figure 3 depicts the overall execution time for all queries for the three configurations, as the number of concurrent queries increases from 10 to 200. For relatively light workload (10 concurrent queries), the overall execution time is reduced by 13.9% in TEM and 16.9% in TEM+. As the number of queries increases, the differences between the three configurations become even more clear. The effective disk I/O bandwidth, which is computed from the number of page requests serviced per second was 8.67MB/sec for the CEM and 13.38MB/sec for the TEM+.

Figure 4 shows the buffer hit ratio as the number of queries increases. Because of the congested query I/O the unsynchronized execution achieves a very poor hit ratio. For the TEM+ the hit ratio increases with the number of concurrent queries and is almost optimal as shown in the Figure.

In Figure 5 we plot the (logical) page requests for 40 queries after they are reordered by the ST and passed to the buffer manager. The ST dynamically aligns requests at the `right` set to exploit spatial locality. These groups are further stacked as shown in the Figure. This I/O pattern is ideal for the LRU policy because of its high time-space locality.

5.2 Experiments with Flow Control

For the following experiment, we implemented the flow-control extensions described in section 4. This new configuration is denoted as TEM+/FC. We set the time-out period W to be 1sec and the sampling period of the Flow Control Thread T to 0.1sec. We then executed 50 concurrent queries and measured the average idle time for each one of them in the tree configurations (CEM,TEM+,TEM+/FC). For the CEM this idle time was 1.7sec on the average for all queries and can be justified from the heavy congestion in the disk for 50 concurrent queries. For TEM+ the average idle time is much higher at 5.8sec

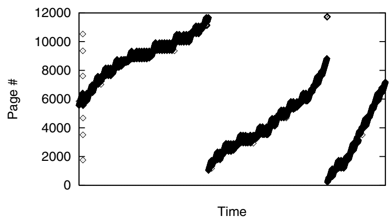


Fig. 5. Requests made from the ST to the Buffer Manager (TEM+)

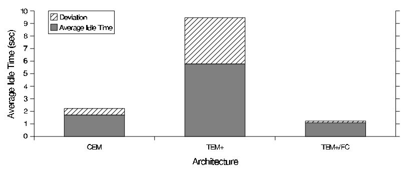


Fig. 6. Idle time comparison

on the average and 21sec in the worst case. Notice that the total execution time is much lower for TEM+ : 91.35sec, v.s. 134.96 sec, i.e. 32.3% lower. The reason that the idle time is higher is because of detached query patches of leaf page requests that are being delayed in the `left` set as described in subsection 4. In Figure 6 we plot the average idle time over all queries along with the computed standard deviation. This graph shows that not only TEM+/FC provides the lowest idle time but also has the smallest standard deviation, which means that it treats all queries fairly.

6 Conclusions

In this paper we argued that conventional index scans and buffering techniques are inadequate for utilizing modern disk hardware and thus fail to support a highly concurrent workload against tree indexes. We showed analytically and through experiments that in an unsynchronized execution, overlapping I/O is only exploited if it occurs within a small time-window. We then introduced the transcurrent execution model (TEM) that exploits intra-parallelism of index scans and dynamically decomposes each query into a set of disjoint query patches. This allows uninterrupted processing of the index, while the disk is serving other I/O requests.

Our experiments demonstrate that the transcurrent query execution results in substantial performance benefits in terms of query execution time, buffer hit ratio and disk throughput. These benefits increase as the workload in the warehouse increases and offer a highly scalable solution to the I/O problem of data warehouses. In addition, TEM can be easily integrated into existing systems; our implementation of ST using posix-threads showed no measurable overhead from the synchronization algorithm and data structures, for 2 up to 500 concurrent queries in a single CPU workstation.

References

1. P. Cao, E. W. Felten, A. R. Karlin, and K. Li. Implementation and Performance of Integrated Application-Controlled File Caching, Prefetching, and Disk Scheduling. *ACM Transactions on Computer Systems*, 14(4):311–343, 1996.

2. C. Y. Chan and Y. Ioannidis. Bitmap Index Design and Evaluation. In *Proceedings of ACM SIGMOD*, pages 355–366, Seattle, Washington, USA, June 1998.
3. S. Chaudhuri and U. Dayal. An Overview of Data Warehousing and OLAP Technology. *SIGMOD Record*, 26(1), September 1997.
4. C.M. Chen and N. Roussopoulos. Adaptive Database Buffer Allocation Using Query Feedback. In *Procs. of VLDB Conf.*, Dublin, Ireland, August 1993.
5. J. Cheng, D. Haderle, R. Hedges, B. Iyer, T. Messinger, C. Mohan, and Y. Wang. An Efficient Hybrid Join Algorithm: A DB2 Prototype. In *Proceedings of ICDE*, pages 171–180, Kobe, Japan, April 1991.
6. H. Chou and D. DeWitt. An Evaluation of Buffer Management Strategies for Relational Database Systems. In *Procs. of VLDB*, Sweden, August 1985.
7. W. Effelsberg and T. Haerder. Principles of Database Buffer Management. *ACM TODS*, 9(4):560–595, 1984.
8. R. Geist and S. Daniel. A Continuum of Disk Scheduling Algorithms. *ACM Transactions on Computer Systems*, 5(1):77–92, 1987.
9. J. Gray. *The Benchmark Handbook for Database and Transaction Processing Systems- 2nd edition.* Morgan Kaufmann, San Franscisco, 1993.
10. J. Gray, P. Sundaresan, S. Englert, K. Baclawski, and P. Weiberger. Quickly Generating Billion-Record Synthetic Databases. In *Proc. of the ACM SIGMOD*, pages 243–252, Minneapolis, May 1994.
11. A. Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. In *Proceedings of the ACM SIGMOD*, Boston, MA, June 1984.
12. Y. Kotidis and N. Roussopoulos. An Alternative Storage Organization for ROLAP Aggregate Views Based on Cubetrees. In *Proceedings of ACM SIGMOD*, pages 249–258, Seattle, Washington, June 1998.
13. R. T. Ng, C. Faloutsos, and T. Sellis. Flexible Buffer Allocation Based on Marginal Gains. In *Procs. of ACM SIGMOD*, pages 387–396, Denver, Colorado, May 1991.
14. C. Nyberg. Disk Scheduling and Cache Replacement for a Database Machine. Master's thesis, UC Berkeley, July 1984.
15. E. J. O'Neil, P. E. O'Neil, and G. Weikum. The LRU-K Page Replacement Algorithm for Database Disk Buffering. In *Proceedings of ACM SIGMOD Intl. Conf. on Management of Data*, pages 297–306, Washington D.C., May 26–28 1993.
16. P. O'Neil and D. Quass. Improved Query Performance with Variant Indexes. In *Proceedings of ACM SIGMOD*, Tucson, Arizona, May 1997.
17. A. Reiter. A Study of Buffer Management Policies for Data Management Systems. Technical Report TR-1619, University of Wisconsin-Madison, 1976.
18. N. Roussopoulos and H. Kang. Principles and Techniques in the Design of *ADMS±. IEEE Computer*, 19(12):19–25, December 1986.
19. N. Roussopoulos, Y. Kotidis, and M. Roussopoulos. Cubetree: Organization of and Bulk Incremental Updates on the Data Cube. In *Proceedings of ACM SIGMOD*, pages 89–99, Tucson, Arizona, May 1997.
20. N. Roussopoulos and D. Leifker. Direct Spatial Search on Pictorial Databases Using Packed R-trees. In *Procs. of ACM SIGMOD*, pages 17–31, Austin, 1985.
21. G. M. Sacco. Index Access with a Finite Buffer. In *Proceedings of 13th International Conference on VLDB*, pages 301–309, Brighton, England, September 1987.
22. A. Shoshani, L.M. Bernardo, H. Nordberg, D. Rotem, and A. Sim. Multidimensional Indexing and Query Coordination for Tertiary Storage Management. In *Proceedings of SSDBM*, pages 214–225, Cleveland, Ohio, July 1999.
23. B. L. Worthington, G. R. Ganger, and Y. N. Patt. Scheduling Algorithms for Modern Disk Drives. In *SIGMETRICS*, Santa Clara, CA, May 1994.

Adaptable Similarity Search Using Vector Quantization

Christian Böhm, Hans-Peter Kriegel, and Thomas Seidl

University of Munich, Oettingenstr. 67, D-80538 Munich, Germany
{boehm,kriegel,seidl}@informatik.uni-muenchen.de

Abstract. Adaptable similarity queries based on quadratic form distance functions are widely popular in data mining applications, particularly for domains such as multimedia, CAD, molecular biology or medical image databases. Recently it has been recognized that quantization of feature vectors can substantially improve query processing for Euclidean distance functions, as demonstrated by the scan-based VA-file and the index structure IQ-tree. In this paper, we address the problem that determining quadratic form distances between quantized vectors is difficult and computationally expensive. Our solution provides a variety of new approximation techniques for quantized vectors which are combined by an extended multistep query processing architecture. In our analysis section we show that the filter steps complement each other. Consequently, it is useful to apply our filters in combination. We show the superiority of our approach over other architectures and over competitive query processing methods. In our experimental evaluation, the sequential scan is outperformed by a factor of 2.3. Compared to the X-tree, on 64 dimensional color histogram data, we measured an improvement factor of 5.6.

1. Introduction

Similarity search in large databases has gained much attention during the last years. A successful approach is provided by feature-based similarity models where appropriate properties of the objects are mapped to vectors of a usually high-dimensional space. The similarity of objects is defined in terms of their distance in the feature space. In addition to the Euclidean distance function which is a very simple but inflexible similarity measure, quadratic forms often provide more appropriate similarity models. Given a positive definite so-called similarity matrix A, the distance of two vectors p and q is defined to be

$$\mathrm{dist}_A(p, q) = \sqrt{(p-q) \cdot A \cdot (p-q)^T}.$$

As the locus of all points p having a distance $\mathrm{dist}_A(p, q) \le \varepsilon$ is an ellipsoid centered around q, the quadratic form-based queries are called *ellipsoid queries* [12].Quadratic form distance functions have been successfully employed for a variety of similarity models in different domains. Examples include the color histogram model for color images in IBM's Query By Image Content (QBIC) system [7, 8], histogram and non-histogram distances for images [13], the shape similarity model for 3D surface segments in a biomolecular database [10], a 2D shape similarity model for clip arts [3], a 3D shape histogram model for proteins [2], or a relevance feedback system [9].

It has been widely recognized that in many application domains, there is not simply one valid measure for the similarity of objects. Instead, the notion of similarity changes with the user's focus of search. This observation has led to the need of user-adaptable similarity models where the user may adapt the similarity distance function to changing application requirements or even personal preferences. As an example, the color histogram approach was extended to become user-adaptable [12]. The query system of [9] based on relevance feedback through multiple examples relies on iterated modifications of the query matrix thus approaching the 'hidden' distance function in the user's mind.

For an efficient query evaluation, feature vectors are often managed in a multidimensional index. Various index structures have been proposed for this purpose [11, 14, 6]. Due to a bunch of problems usually called the '*curse of dimensionality*', even specialized index structures deteriorate with increasing dimension of the feature space. It has been shown [15] that in many situations, depending on parameters such as the dimension, the data distribution and the number of objects in the database, indexes often fail to outperform simpler evaluation methods based on the sequential scan of the set of feature vectors. The solution proposed by Weber, Schek and Blott is therefore an improvement of the sequential scan by quantizing the feature vectors, called *VA-file* [15]. The general idea of the VA-file is to store the features not with the full precision of e.g. 32 bit floating point values but to use a reduced number of bits. For this purpose an irregular, quantile based grid is laid over the data space. Following the paradigm of multistep query processing, candidates produced by this filter step are exactly evaluated by a subsequent refinement step. The gain of the lossy data compression is a reduced time to load the feature vectors from secondary storage. Queries using Euclidean distance metric can be accelerated by factors up to 6.0 with this technique, if the search is I/O bound.

While the VA-file is an excellent choice for Euclidean and weighted Euclidean distance metric, several problems come up when the similarity measure is a quadratic form distance function. First, the scan based processing of ellipsoid queries is typically CPU bound, because the determination of the distance function has quadratic time complexity with respect to the dimension of the feature space. Therefore, a pure acceleration of the I/O time will hardly improve the answer time. A second problem is that it is generally difficult to determine the distance between the query point and the grid approximation of the feature vector if the distance measure is a general (i.e. not iso-oriented) ellipsoid. The reason is that the intersection point between the ellipsoid and the approximation cell can only be determined by expensive numerical methods.

Our solution to these problems is a series of three filter steps with increasing evaluation cost and decreasing number of produced candidates. These filter steps approximate both, the query ellipsoid and the grid cells at different levels of accuracy. Our first filter step performs a new kind of approximation of the query ellipsoid, the approximation by an axis-parallel ellipsoid. As it corresponds to a weighted euclidean distance calculation, this approximation can be evaluated very efficiently for vector approximations. Unfortunately the selectivity of this filter step is for some query ellipsoids not good enough as desired. The selectivity can be improved by large factors, however, if the first filter is followed by a novel technique approximating the grid cells. This technique determines the distance between the query point and the center of the grid cell, additionally considering a maximum approximation error. The maximum approximation error represents the maximum (ellipsoid) distance between the center point of the grid cell and its most distant corner point. This will be our last filter step. As the determination of the maximum approximation error has quadratic time complexity, we propose as an intermediate filter step another conservative approximation of the maximum approximation error which can be determined in linear time.

The benefits of our technique are not limited to the extension of the VA-file approach. Our filter steps can always be applied when ellipsoid distances to rectilinear rectangles have to be efficiently estimated. For instance, the page regions of most index structures are rectangular. Therefore, processing the directory pages of a multidimensional index structure can be improved by our technique. Recently, the idea of vector approximation of grid cells was also applied to index based query processing in the so-called IQ-tree [4]. Our multi-step query processing architecture enables efficient processing of ellipsoid queries in the IQ-tree. As one concrete example, however, we put our focus in this paper to processing of adaptable similarity queries by the VA-file. Out of the focus of this paper is dimensionality reduction [7, 12]. Reduction techniques based on the principal components analysis, fourier transformation or similar methods can be applied to our technique and all competitive techniques as a preprocessing step.

The rest of this paper is organized as follows: In section 2 (related work) we introduce various techniques for evaluating ellipsoid queries and the vector quantization. In section 3 we describe our novel techniques for approximating grid cells by minimum bounding ellipsoids in quadratic time, our linear approximation of the minimum bounding ellipsoid and our new approximation of the query ellipsoid by an iso-oriented ellipsoid. Section 4 is dedicated to the filter architecture. Here, we demonstrate in what situation what filter has particular strengths and how our three filters complement each other. Thus, it is useful to combine our three proposed filters, and each of the filters yields additional performance gains. The most efficient order of filter steps is also determined. Finally, we perform a comprehensive experimental evaluation in section 5.

2. Related Work

The evaluation of ellipsoid queries requires quadratic time in the data and directory nodes of the index. In order to reduce this effort, approximation techniques for ellipsoid queries have been developed that have a linear complexity thus supporting linear filter steps [1]. Conservative approximations of query ellipsoids such as the minimum bounding sphere, the minimum bounding box or the intersection of both guarantee no false dismissals for range queries. As a more general concept, equivalent lower-bounding distance functions were presented that guarantee no false dismissals for (*k*-)nearest neighbor queries in addition to similarity range queries.

Sphere Approximation. The *greatest lower-bounding sphere distance function* $d_{\mathrm{glbs}(A)}$ of an ellipsoid is a scaled Euclidean distance function defined as follows. Let A be a similarity matrix, and $w^2_{\min}$ be the minimum eigenvalue of the matrix A, then

$$d_{\mathrm{glbs}(A)}(p, q) = w_{\min} \cdot |p - q| .$$

In [1], it is shown that the function $d_{\mathrm{glbs}(A)}$ is the greatest scaled Euclidean distance function that lower bounds the ellipsoid distance function d_A, i.e. for all $p, q \in \Re^d$:

$$d_{\mathrm{glbs}(A)}(p, q) \leq d_A(p, q) .$$

Note particularly that $d_{\mathrm{glbs(A)}}$ is well-defined since the eigenvalues of the positive definite similarity matrix are positive.

In addition to the minimum bounding sphere, also the minimum bounding box approximation has been generalized to an equivalent distance function:

Box Approximation. The *greatest lower-bounding box distance function* $d_{\mathrm{glbb}(A)}$ of an ellipsoid is a weighted maximum distance function. For any similarity matrix A, the inverse A^{-1} exists, and we define:

$$d_{\mathrm{glbb}(A)}(p, q) = \max\left\{ \frac{|p_i - q_i|}{\sqrt{(A^{-1})_{ii}}}, i = 1 \ldots d \right\}$$

The function $d_{\mathrm{glbb}(A)}$ is the greatest weighted maximum distance function that lower bounds the ellipsoid distance function d_A, i.e. for all $p, q \in \Re^d$:

$$d_{\mathrm{glbb}(A)}(p, q) \leq d_A(p, q)$$

Vector Quantization. The general idea of the VA-file [15] is to store the components of the feature vectors not with full (e.g. 32 bit) precision, but with a reduced precision of e.g. 5-6 bits. For this purpose, the data space is partitioned by an irregular grid which is determined according to the quantiles of the components of the feature vectors. Instead of the exact position of each point in the database, only the grid cell is stored in the VA-file. Scanning the reduced VA-file instead of the file containing the full precision data saves I/O time proportional to the compression rate. If a cell is partially intersected by the query region, it cannot be decided whether the point is in the result set or not. In this case, an expensive lookup to the file containing the full precision point data is due. Therefore, this approach corresponds to the paradigm of multistep query processing where the distance to the grid cell is a lower bound of the distance to the point. For Euclidean distance measures, the distance to a given cell can be determined very efficiently by precomputing the squared distances between the query point and the quantiles. Therefore, such a distance calculation can even be slightly cheaper than determining the distance between two points.

Recently, the idea of vector quantization was adopted to index based query processing. In the IQ tree [4] a separate regular grid is laid over each page region. Each data page has two representations: one containing the compressed (quantized) representation of the data points and the second containing the points in full precision. The optimal precision for the compressed data page as well as an optimal page schedule using a "fast index scan" operation are determined according to a cost model [5].

3. Ellipsoid Queries on Quantized Vectors

Compared to axis-parallel ellipsoids and spheres, it is much more difficult to determine whether a general ellipsoid intersects a rectangle. The reason for this difference is depicted in figure 1: For the axis-parallel ellipsoid, the closest point to the center of the ellipsoid can be easily determined by projecting the center point. In contrast, the general ellipsoid may in-

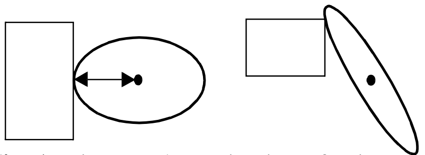


Fig. 1. Distance determination of axis-parallel and general ellipsoids to a rectangle

tersect the rectangle at its upper right corner although the center of the ellipsoid is underneath the lower boundary of the rectangle. This is impossible for unskewed ellipsoids and this property facilitates distance determination for Euclidean and weighted Euclidean metric.

The exact ellipsoid distance between a point and a rectangle can therefore only be determined by a time consuming numerical method [12], which is not suitable as a filter step. An approximation of the exact distance is needed which fulfills the lower bounding property. Our idea for this filter step is not to approximate the query (which is done in another filter step) but to approximate the rectangle. Unfortunately, the most common approximation by minimum bounding spheres suffers from the same problem as the rectangle itself: It is numerically difficult to determine the ellipsoid distance between the query point and the sphere. A suitable approximation, however, is an ellipsoid which has the same shape as the query ellipsoid, i.e. the same lengths and directions of the principal axes. We will show in the following that this approximation can be determined with low effort and that it can be used to define a lower bounding of the exact ellipsoid distance.

3.1 The Minimum Bounding Ellipsoid Approximation (MBE)

We want to determine the minimum ellipsoid enclosing a given rectangle R which is geometrically similar to the query ellipsoid. Due to the convexity of an ellipsoid, the rectangle is contained in the ellipsoid if all of its corners are contained. But we cannot investigate all 2^d corners of the rectangle to determine the minimum bounding ellipsoid. We need to determine the corner of the rectangle with maximum (ellipsoid) distance from the center of the rectangle. The following lemma will show that it is the corner which is closest to the eigenvector corresponding to the largest eigenvalue.

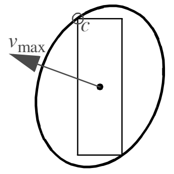


Fig. 2. Minimum bounding ellipsoid

Lemma 1. A rectangle is contained in an ellipsoid with the same center iff the corner which is closest to the eigenvector corresponding to the largest eigenvalue is contained in the ellipsoid.

Proof. Let $A = VW^2V^T$ with $VV^T = \mathrm{Id}$, $W^2 = \mathrm{diag}(w_i^2)$ be the diagonalization of A where V_i denotes the eigenvector corresponding to the eigenvalue w_i^2, and let c be any of the corners of a rectangle centered at the origin. To compute the ellipsoid distance of the corner c to the origin 0, consider the following transformation: $c \cdot A \cdot c^T = c \cdot VW^2V^T \cdot c^T = (cV) \cdot W^2 \cdot (cV)^T = (cVW) \cdot (cVW)^T$. These terms are quadratic forms for the matrices A, W^2 and Id applied to the vectors c, cV and cVW, respectively, i.e. $d_A(c, 0) = d_{VW^2V^T}(c, 0) = d_{W^2}(cV, 0) = d_{Id}(cVW, 0) = |cVW|$.

Due to the orthogonality of V, all the original corner points c and transformed corners cV have the same Euclidean distance $|c| = |cV|$ to the origin. Only after the iso-oriented weighting by W, i.e. stretching the axis e_i by the weight w_i, the lengths $|cVW|$ differ. Let $e_{\max}$ be the axis corresponding to the maximum weight $w_{\max} = \max\{w_i\}$. The maximum value of $|cVW|$ is obtained for the corner cV that is closest to the axis $e_{\max}$, i.e. the angle between cV and $e_{\max}$ is minimal:

$$\mathrm{angle}(cV, e_{\max}) = \mathrm{acos}\frac{cV \cdot e_{\max}}{|cV| \cdot |e_{\max}|} = \mathrm{acos}\frac{c \cdot v_{\max}}{|c|}$$

Proof. At all, the corner $c_{\max}$ having the maximal distance value $d_A(c_{\max}, 0)$ is the corner that is closest to $v_{\max}$, i.e. the eigenvector corresponding to the maximal eigenvalue. ❑

The corner closest to the largest eigenvector can be determined in linear time by considering the sign of each component of the eigenvector: If the sign is positive, we take the upper boundary of the rectangle in this dimension, otherwise the lower boundary. If a component is 0, it makes no difference whether we take the lower or upper boundary because both corners yield equal distance:

```
Algorithm $A_1$:
        closest (e: vector [d], l: vector [d], u: vector[d]): vector[d]
            for i:=1 to d do
                if e[i] >= 0 then closest[i] := (u[i] - l[i])/2 ;
                              else closest[i] := (l[i] - u[i])/2 ;
```

If c is the closest corner determined by algorithm A_1, the distance d_{MBE} is the maximum ellipsoid distance between the center of the grid cell and an arbitrary point in the cell where d_{MBE} is given:

$$d_{\mathrm{MBE}} = \sqrt{c \cdot A \cdot c^T}$$

For the distance between an arbitrary point p in the cell with center point p_c and the query point q, the following inequality holds:

$$d_A(p, q) \geq d_A(p_c, q) - d_{\mathrm{MBE}}$$

This is simply a consequence of the triangle inequality which is valid for ellipsoid distances. The computation of d_{MBE} has a quadratic time complexity, because the multiplication of the matrix and a vector is quadratic. In the next section, we will develop an additional approximation which is an upper bound of d_{MBE}. Given a query object and a database object, these approximations can be evaluated in time linear in the number of dimensions.

3.2 The Rhomboidal Ellipsoid Approximation (RE)

The computation of the approximated distance $d_A(p_c, q) - d_{\mathrm{MBE}}$ is quadratic in the number of the dimensions. This is already much better than determining the exact distance between query point and approximation cell which can only be done by expensive numerical methods. In order to further reduce the computational effort, we develop in this section an additional filtering step which is *linear* in the number of dimensions. It would be possible to avoid the computation of d_{MBE} for each item stored in the VA-file, if we would determine $d_{\mathrm{MBE,max}}$ i.e. the ellipsoid approximation of the largest cell stored in the VA-file. Unfortunately, the sizes of the grid cells vary extremely for real data sets due to the quantile concept. Therefore, $d_{\mathrm{MBE,max}}$ is a bad approximation of most of the grid cells. The size of an individual cell must be considered in a suitable way.

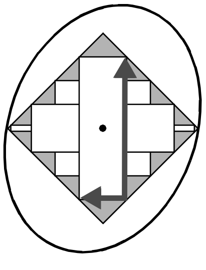


Fig. 3. Rhomb. ellips. app.

Our way to consider the cell size is based on the sum of the side lengths of the cell. This sum can obviously be determined in O(d) time. The idea is to determine in a preprocessing step an ellipsoid which contains all cells having a standardized sum of the side lengths. If the actual sum of the side lengths of a grid cell is larger or smaller than this standardized value, the ellipsoid is scaled, which is an operation of constant time complexity.

As depicted in figure 3 the hull of all rectangles having a sum $s = \Sigma_{i=1\ldots d}\, s_i$ of the extensions s_i of s forms a rhomboid with diameter s or "radius" $s/2$. The corners of the rhomboid are the unit vectors, each multiplied with $+s/2$ and $-s/2$, respectively. Due to convexity and symmetry, the minimum bounding ellipsoid of the rhomboid touches at least two of the corners, which are on opposite sides with respect to the origin.

This observation leads to the following lemma:

Lemma 2. For a similarity matrix $A = [a_{ij}]$, the minimum bounding ellipsoid over all cells that have a sum $s = \Sigma_{i=1\ldots d}\, s_i$ of extensions has a distance value of

$$d_{\mathrm{RE}}(s) = \frac{s}{2} \cdot \max\,\{\sqrt{a_{ii}},\, i = 1\ldots d\}$$

Proof. The minimum bounding ellipsoid touches the rhomboid at the corners. Due to symmetry, we can restrict ourselves to the positive unit vectors scaled by $s/2$, i.e. $e_i \cdot s/2$, $i = 1\ldots d$. The rhombus is touched at the unit vector which has the largest ellipsoid distance from the center, because all other corners must not be outside of the ellipsoid, and this distance equals to $d_{\mathrm{RE}}(s)$.

Proof. The ellipsoid distance between the i-th unit vector e_i and the origin 0 is $d_A(e_i, 0) = \sqrt{e_i \cdot A \cdot e_i^T} = \sqrt{a_{ii}}$, and the maximum over the dimensions $i = 1\ldots d$ is the square root of the maximal diagonal element of A, i.e. $\max\{d_A(e_i, 0)\} = \max\{\sqrt{a_{ii}}\}$. Scaling by $s/2$ is commutative with d_A and with the maximum operator and, hence, the proposition holds. ❑

With lemma 2, we can estimate the maximum distance between a point approximated by a grid cell and the center of the grid cell. In contrast to the distance d_{MBE} introduced in section 3.1 this bound can be determined in linear time. It is an upper bound of d_{MBE}, as can be shown as follows:

Lemma 3. $d_{\mathrm{RE}}(s)$ is an upper bound of d_{MBE}.

Proof. Let Rh(s) be the locus of all points p where $\Sigma_{i=1\ldots d} |p_i| \leq s/2$. By lemma 2, we know that for each point $p \in$ Rh(s) the ellipsoid distance to the origin $d_A(p, o)$ is at most $d_{\mathrm{RE}}(s)$. Let C(s) be an arbitrary grid cell centered at the origin having a side sum of s. For every point

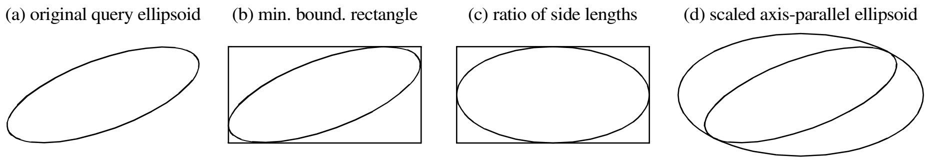

(a) original query ellipsoid
(b) min. bound. rectangle
(c) ratio of side lengths
(d) scaled axis-parallel ellipsoid

Fig. 4. Construction of the minimum axis-parallel ellipsoid for a given general ellipsoid

$q \in \mathrm{C}(s)$ we know that $\Sigma_{i=1\dots d}|q_i| \le s/2$. It follows that $d_A(q, o) \le d_{\mathrm{RE}}(s)$. As d_{MBE} is the maximum ellipsoid distance of all points $q \in \mathrm{C}(s)$, we have

$$d_{\mathrm{MBE}} = \max\{d_A(q, o), q \in C(s)\} \le d_{\mathrm{RE}}(s) \qquad ❑$$

Since the rhomboidal ellipsoid approximation is less exact than the minimum bounding ellipsoid approximation, it is likely to yield a worse filter selectivity (i.e. a higher number of candidates). However, it can be determined by far faster. We will see in section 4 that the determination of the rhomboidal ellipsoid approximation causes almost no extra cost compared to the MBE approximation, but avoids many expensive evaluations of the MBE approximation. Therefore, it is intended to apply the combination of these two filters.

3.3 Axis-Parallel Ellipsoid Approximation

In this section, we propose an additional filtering step which now approximates the query ellipsoid, not the grid cells. In the next section, we will show that it is beneficial to approximate both, queries and grid cells in separate filtering steps, because the two different approximation techniques exclude quite different false hits from processing, and, therefore, the combination of both methods yields a much better selectivity than either of the methods alone. We again propose an approximation technique which is particularly well suited for grid based quantization. In the VA-file (as well as in the IQ-tree and any grid based method) the computation of weighted Euclidean distances is very simple. Basically, it makes no difference whether to determine a weighted or an unweightet Euclidean distance. Therefore, we approximate the general query ellipsoid by the minimum axis-parallel ellipsoid that contains the query ellipsoid.

The axis parallel ellipsoid is constructed in two steps. In the first step, the ratio of the side lengths of the ellipsoid is determined. It corresponds to the ratio of the side lengths of the minimum bounding rectangle of the ellipsoid. In the second step, the axis-parallel ellipsoid is scaled such that it is a minimum bound of the original query ellipsoid. This is done by scaling both the original and the axis-parallel ellipsoid non-uniformly such that the axis-parallel ellipsoid becomes a sphere. The matrix corresponding to the scaled original query ellipsoid is then diagonalized to determine the smallest eigenvalue w_{min}. This eigenvalue corresponds to the factor by which the axis-parallel ellipsoid must be (uniformly) scaled such that it minimally bounds the original query ellipsoid. The process of constructing the minimum bounding axis-parallel ellipsoid is shown in figure 4.

Lemma 4. Among all axis-parallel ellipsoids our solution is a minimum bounding approximation of the query ellipsoid.

Proof. Follows immediately from the construction process. ❑

4. Analysis of the Filter Architecture

In this section, we will develop a filter architecture for adaptable similarity queries which is adjusted to the VA-file and other grid-based quantization methods and which optimizes query processing based on vector quantization. We have proposed three new approximation techniques for this purpose in section 3, the minimum bounding ellipsoid approximation, the rhomboidal ellipsoid approximation, and the minimum bounding axis-parallel ellipsoid. The first and second technique approximates the quantization cells by ellipsoids. The shape of these approximating ellipsoids is defined by the similarity matrix A. In contrast, the third technique approximates the query ellipsoid by an axis-parallel ellipsoid which corresponds to a weighted euclidean distance calculation. We are now going to discuss the various approximation techniques according to their most relevant parameters, computational effort and filter selectivity.

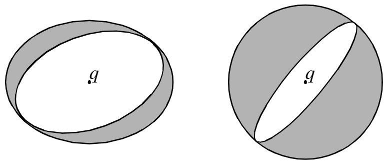


Fig. 5. Selectivity of axis-parallel approx.

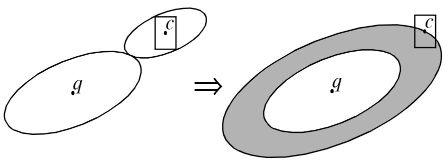


Fig. 6. Selectivity of the MBE approx.

4.1 Basic Approximations

The axis-parallel approximation of the query ellipsoid requires the determination of the smallest eigenvalue which is done in a preprocessing step when reading the similarity matrix for the query. Although this operation has a bad time complexity, preprocessing operations are still considered to be negligible as we generally assume $d \ll N$. Once the smallest eigenvalue is determined, we have only a linear effort O(d) per considered object. As discussed in section 2, the VA-file allows a particularly efficient determination of the weighted Euclidean distance between the query point and the approximation cell, because the squared differences between the quantiles and the query point can also be precomputed outside the main loop iterating over all objects. Therefore, d additions are the only floating point operations performed per object. To assess the selectivity of this filter step, refer to figure 5. Here, we distinguish between queries which are more or less axis-parallel. The left side of figure 5 shows an ellipsoid which is very well approximable. In contrast, the right side presents an ellipsoid with an angle close to 45°. The axis-parallel approximations are also depicted. Underlaid in gray is the volume which represents the approximation overhead. Objects from this area are candidates produced by the filter step, however, they are no hits of the similarity query. The smaller the gray overhead volume, i.e. the more axis-parallel the original query ellipsoid, the better is the filter selectivity. Assuming a uniform distribution of points in the data space, we would have a few percent overhead for the left query ellipsoid, but several hundred percent overhead for the right query ellipsoid. For less axis-parallel original ellipsoids, the filter selectivity is not sufficient.

The MBE approximation of the grid cells requires the highest CPU time of all considered filters. To determine whether a grid cell contains a candidate point or not, two ellipsoid distance computations must be performed: First, the distance between the point and the center of the cell and second, the distance between the center of the cell and the corner closest to the “largest” eigenvector have to be computed. To assess the selectivity of this filter, we apply a problem transformation. For our analysis, we assume that all grid cells have identical shapes and, therefore, the minimum bounding ellipsoids are all identical. This allows us to transform the problem in such a way that we add the range of the MBE to the range of the query, as depicted in figure 6: In the left part, the cell is a candidate if the query ellipsoid intersects the MBE approximation of the cell. In the right part, we have transformed the query ellipsoid into a larger ellipsoid, where the range is the sum of the two ranges of the query and the MBE. The cell is a candidate whenever the center of the cell is contained in the enlarged ellipsoid. This concept of transformation is called Minkowski sum [5]. Again, in the right part of figure 6, the approximation overhead is marked in gray. Using this approximation, the overhead heavily depends on the size of the MBE approximation.

The RE approximation of the grid cells is very similar in both, computation time and selectivity. In contrast to the MBE approximation, this techniques requires only one distance calculation: The distance between the query point and the center of the grid cell. That means, one application of this filter step causes about half the expenses compared to one application of the MBE filter, assuming that all other cost are negligible. For this technique, we have a similar selectivity assessment as for the MBE approximation depicted in figure 6. The only difference is that the ellipsoid approximating the cell is not minimal. Therefore, also the Minkowski sum and the implied approximation overhead are larger than the MBE overhead in figure 6.

Last but not least, we have to consider the cost of the refinement step. The computational cost is limited to a single ellipsoid distance computation, which is even less expensive than an application of the MBE filter and comparable to the cost of the RE filter. In contrast to all described filter steps, the refinement step works on the exact coordinates of the data point and not on the grid approximation. Therefore, the point must be loaded from background storage which usually causes one costly seek operation on the disk drive. The refinement step is the most expensive step unless the distance calculation cost exceed the cost for a random disk access. Table 1 summarizes the results of this section.

4.2 Combined Approximations

Table 1: Cost and selectivity of filters

Technique	**Cost**	**Selectivity**
axis parallel ellip. approx.	$O(d)$	low
MBE approx.	$2 \cdot O(d^2)$	fair
rhomb. ell. ap.	$1 \cdot O(d^2)$	medium
exact distance	$1 \cdot O(d^2) + 1$ random I/O	exact

In this section, we show that the combined application of the developed filters is useful and results in systematic performance advantages compared to the application of a single filter. First we consider the combined application of the MBE filter and the rhomboidal ellipsoid filter. We first show that the combined application of MBE and RE does not cause any considerable extra cost compared to the application of the MBE filter alone. The application of the MBE filter requires the distance calculation between the query point and the center of the cell, and the distance calculation between the cell center and a cell corner. The RE filter requires the distance calculation between the query point and the center of the cell only. All other cost are negligible. If we first apply the RE filter and then the MBE filter, the MBE filter may reuse the distance evaluated by the RE filter. The combined application performs one distance calculation for all points and a second calculation for those points which are not excluded by the RE filter. Whether the filter combination is more efficient than the application of the MBE filter alone depends on the selectivity of the two filters. We know that the MBE filter is at least as selective as the RE filter. However, if the MBE filter does not save additional I/O accesses compared to the RE filter, the additional distance calculations lead to a performance deterioration. Since I/O accesses are usually much more expensive than distance calculations, the breakeven point can be reached when the MBE filter is only slightly more selective than the RE filter. Our experimental evaluation will show, that the selectivity of the MBE filter is significantly better. It is beneficial to apply the RE filter first and then the MBE filter, because the cost of the RE filter are lower.

Next we are going to show that it is beneficial to combine the axis-parallel approximation and the MBE approximation. In figure 8, we can compare the selectivities of the two filters. In contrast to the previously described combination, the filters are not lower bounds of each other, but access rather different parts of the data space. The area of the main overhead of the axis-parallel approximation is where the query ellipsoid is narrow (depicted on the left side of fig. 8). The MBE approximation, however, yields most of its overhead at the front and the bottom of the ellipsoid, where the axis-parallel approximation yields no overhead at all. Therefore, the combined filter yields a dramatically improved selectivity compared to the axis-parallel approximation as well as compared to the MBE filter, as depicted in the rightmost illustration in figure 8. As the axis-parallel approximation is by far cheaper than the MBE filter, it is natural to apply the axis-parallel approximation first. For the order of these filters, we consider the cost of the filter step per evaluated point:

$$C_{\text{axis-par}} << C_{\text{RE}} << C_{\text{MBE}} << C_{\text{exact}}.$$

Therefore, we apply the axis-parallel approximation as the first filter step. The second filter step is the RE filter. The MBE approximation is the third filter followed by the refinement step.

5. Experimental Evaluation

To evaluate our filter techniques experimentally, we have implemented a VA-file extension with our four-step query processing architecture. Our implementation in C was tested on a HP C-160

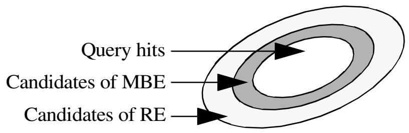


Fig. 7. Combination of RE and MBE filters

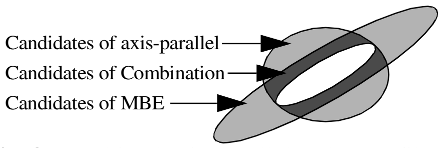


Fig. 8. Combined axis-parallel and MBE filter

workstation under the HPUX operating system. Both, vector quantizations and exact point data were stored on the same disk drive with an average seek time of 8 ms and a transfer rate of 4 MByte/s for continuous data transfer. The vector approximations were scanned in large blocks of 1 MByte to minimize the influence of rotational delays or disk arm repositionings between subsequent I/O requests.

Our reference application is a similarity search system for color images which allows a user-adaptable specification of the similarity measure based on color histograms [12]. Our data set contains 112,363 color images, each represented by a 64-dimensional color histogram. We separated our file into a large data file and a smaller query file by random selection. On our hardware, an evaluation of an ellipsoid distance calculation needs 60 µs and a Euclidean distance calculation requires about 0.5 µs. Our implementation performs ellipsoid range queries. Unless otherwise mentioned our experiments determine the 2 nearest neighbors of 10 query points.

We generate the similarity matrices $A = [a_{ij}]$ by the formula $a_{ij} = \exp(-\sigma(d(i,j)/d_{\max})^2)$ from [8] where $d(i,j)$ denotes the cross-similarity of the colors i and j. Since the adaptable similarity model supports the modification of local similarities, we introduce three additional parameters σ_r, σ_g, σ_b and define $d(i,j) = \mathrm{sqrt}(\sigma_r \Delta r^2 + \sigma_g \Delta g^2 + \sigma_b \Delta b^2)$ in the RGB color space. For our queries, we use five different parameter settings for the tuple $(\sigma, \sigma_r, \sigma_g, \sigma_b)$, namely M_1 (1, 100, 1, 1), M_2 (1, 1, 1, 100), M_3 (100, 1, 1, 1), M_4 (100, 100, 1, 1, 1) and M_5 (1, 1, 1, 1).

In the previous sections, we have postulated several claims which require an experimental validation beyond the actual proof of superiority over competitive techniques. The most important claims to justify our four-step query processing architecture were

- the superiority of the combination of the axis-parallel approximation and the MBE filter and
- the benefit of the second filter step (RE approximation)

In our first experiment (cf. figure 9) we measure the selectivities of the axis-parallel approximation and the MBE filter both, separately as well as combined. As already pointed out in section 4, the quality of the axis-parallel approximation depends on the orientation of the query ellipsoid. Our ellipsoid M_1 has a bad selectivity (14,473 candidates equivalent to 12.5% of all points). Only ellipsoid M_3, which is almost a sphere, yields a satisfactory selectivity of 3.3 candidates which means an overhead of 1.3 candidates for 2 neighbors. The selectivity of the MBE filter, applied separately, is with 457 candidates for ellipsoid M_1 by far better than the selectivity of the axis-parallel approximation, but not satisfactorily. For the queries M_3, M_4 and M_5, the selectivity of the MBE filter is partly better, partly worse than that of the axis-parallel approximation. The combination of both filters, however, outperforms the separate applications of the approximations by large factors (up to 176 compared to the axis-parallel approximation and up to 5.6 compared to the MBE filter). Only for ellipsoid M_3 where the axis-parallel approximation is almost optimal the combination does not yield further selectivity improvements.

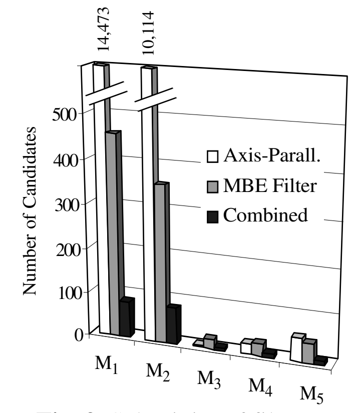


Fig. 9. Selectivity of filters.

For the ellipsoids M_1 and M_2, the RE filter yields around 12,000 candidates, and thus a selectivity in the same order of magnitude as the axis-parallel approximation. Now one may wonder whether the second filter step is useful at all. Moreover, for ellipsoid M_5, the number of candidates of the RE filter is even by a factor of 250 worse than the axis-parallel approximation. Our next experiment depicted in figure 10, however, shows that the RE filter, like the MBE filter, is beneficially combinable with the axis-parallel approximation. As we have pointed out in section 4, the RE approximation as an additional filter step is justified if it yields fewer candidates than the axis-parallel approximation but substantially more candidates than the MBE filter (otherwise, the MBE filter would be unnecessary). For the first two ellipsoids every filter step reduces the candidate set approximately to 10%. Note that the scale in this figure is logarithmic. In ellipsoid M_1, for instance, the axis-parallel approximation reduces the candidate set from 112,000 to 14,473 (12.9%). The second filter step reduces this set further to 1,458 (10.1% of 14,473) and the third filter step to 82

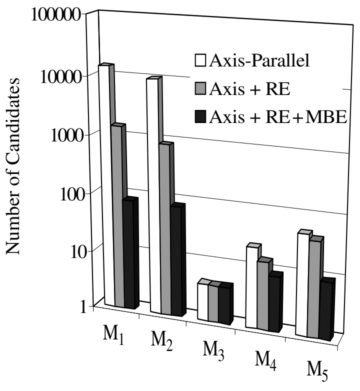


Fig. 10. Comparing architectures (# candid.)

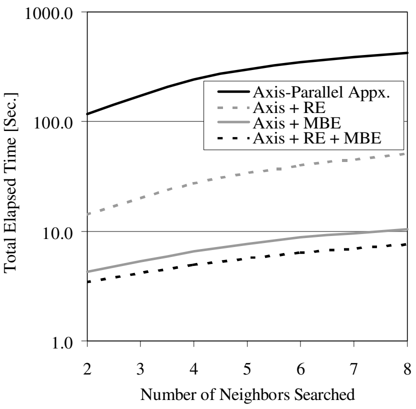
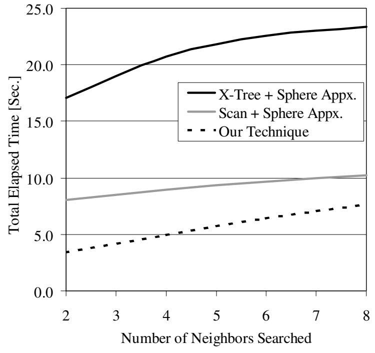


Fig. 11. Comparing architectures (time)

(5.6% of 1,458). Even for ellipsoid M_5, for which the selectivity of the RE filter yielding 13,216 candidates (cf. figure 10) is extremely bad, the combination of the axis-parallel approximation and the RE filter clearly outperforms the axis-parallel approximation by a factor 1.2.

The next experiment demonstrates the impact of the various numbers of candidates on the overall runtime of a query. Therefore, we compare the runtime of our four-step architecture with a two-step architecture (axis-parallel approximation and refinement step) and also with two architectures consisting of three steps: (1) axis-parallel approximation, RE approximation, refinement step; (2) axis-parallel approximation, MBE approximation, refinement step. Figure 11 shows the results of our experiments using ellipsoid M_1 in logarithmic scale. In this figure the selectivity is varied such that the number of retrieved neighbors ranges from 2 to 8. The performance of the two-step architecture performing merely the axis-parallel approximation upon the VA file is very bad (119 to 234 seconds of total time). It is clearly outperformed by the three-step and four-step architectures using our new types of cell approximation in combination with the axis-parallel approximation. The architecture with the additional rhomboidal ellipsoid filter requires between 14 and 51 seconds and thus is up to 8.5 times faster. The improvement factor of the three-step architecture with the MBE filter is even higher, up to 28.3. The constantly best performance shows our four-step architecture: 3.5 to 7.7 seconds total elapsed time with an improvement factor of 34 over the two-step architecture.

In a last series of experiments we compare our new technique (four-step architecture) with two competitive techniques, the sequential scan and the X-tree index. Both competitive techniques are



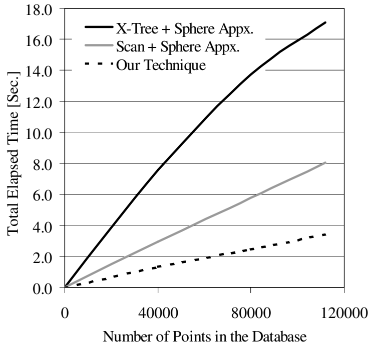


Fig. 12. Comparison with varying selectivity (l.) and scalabiltiy (r.)

allowed to use the sphere approximation as a filter step. In the left diagram of figure 12, we measure again the total processing time with varying selectivity. The sequential scan which requires between 8.1 and 10.2 seconds is outperformed by factors between 1.3 and 2.3, and the X-tree index (17.1 to 23.4 sec.) is outperformed by factors between 3.0 and 4.9. The right diagram shows the same experiment with varying database size. With increasing database size, the improvement factor over the X-tree slightly increases from 4.7 (40,000 points) to 5.0 (112,000 points) while the improvement factor over the sequential scan remains constant at 2.3.

6. Conclusions

In this paper, we have proposed three new approximation techniques to cope with the problem of efficiently processing user adaptable similarity queries on quantized vectors. The first and second filter approximate the cells of the quantization grid by ellipsoids with the same principal axes as the query ellipsoids and enable us to exploit the triangle inequality. The MBE approximation determines the minimum bounding ellipsoid for each quantized vector and, hence, requires a quadratic time complexity. In contrast, the RE filter requires linear time in query processing. Our third new filter technique approximates the query which is a general ellipsoid (corresponding to a quadratic form distance) by its minimum bounding axis-parallel ellipsoid. Axis-parallel ellipsoids correspond to weighted Euclidean distances which can be evaluated with particular efficiency on grid based query processing techniques such as the VA-file or the IQ tree. We propose a multistep query processing architecture with three filter steps (our new axis-parallel approximation and our new filters RE and MBE) and show the superiority over architectures with fewer filter steps and over competitive techniques theoretically as well as experimentally. Our anylysis demonstrates that our filters complement each other. Hence, it is useful to combine our three filters, and we determine a suitable order of the filter steps. In our experimental evaluation, the sequential scan is outperformed by a factor of 2.3. Compared to the X-tree on 64 dimensional color histogram data, we measured an improvement factor of 5.7. For our future work, we plan to integrate our new approximation techniques into the IQ-tree.

References

1. Ankerst M., Braunmüller B., Kriegel H.-P., Seidl T.: *Improving Adaptable Similarity Query Processing by Using Approximations.* Proc. 24th Int. Conf. on Very Large Data Bases (VLDB), 1998, 206-217.
2. Ankerst M., Kastenmüller G., Kriegel H.-P., Seidl T.: *3D Shape Histograms for Similarity Search and Classification in Spatial Databases*. Int. Symp. on Spatial Databases (SSD), LNCS 1651, 1999, 207-226.
3. Ankerst M., Kriegel H.-P., Seidl T.: *A Multi-Step Approach for Shape Similarity Search in Image Databases.* IEEE Transactions on Knowledge and Data Engineering (TKDE) 10(6), 1998, 996-1004.
4. Berchtold S., Böhm C., Jagadish H. V., Kriegel H.-P., Sander J.: *Independent Quantization: An Index Compression Technique for High-Dimensional Data Spaces*, Int. Conf. on Data Engineering (ICDE), 2000.
5. Berchtold S., Böhm C., Keim D., Kriegel H.-P.: *A Cost Model For Nearest Neighbor Search in High-Dimensional Data Space*. ACM PODS Symposium on Principles of Database Systems, 1997.
6. Berchtold S., Keim D., Kriegel H.-P.: *The X-tree: An Index Structure for High-Dimensional Data.* Proc. 22nd Int. Conf. on Very Large Data Bases (VLDB), 1996, 28-39.
7. Faloutsos C., Barber R., Flickner M., Hafner J., Niblack W., Petkovic D., Equitz W.: *Efficient and Effective Querying by Image Content.* Journal of Intelligent Information Systems, Vol. 3, 1994, 231-262.
8. Hafner J., Sawhney H. S., Equitz W., Flickner M., Niblack W.: *Efficient Color Histogram Indexing for Quadratic Form Distance Functions.* IEEE Trans. on Pattern Analysis and Machine Intelligence (PAMI) 17(7), 1995, 729-736.
9. Ishikawa Y., Subramanya R., Faloutsos C.: *MindReader: Querying Databases Through Multiple Examples*. Proc. 24th Int. Conf. on Very Large Data Bases (VLDB), 1998, 218-227.
10. Kriegel H.-P., Seidl T.: *Approximation-Based Similarity Search for 3-D Surface Segments*. GeoInformatica Int. Journal, Vol. 2, No. 2. Kluwer Academic Publishers, 1998, 113-147.
11. Lin K., Jagadish H. V., Faloutsos C.: *'The TV-Tree: An Index Structure for High-Dimensional Data.* VLDB Journal 3(4), 1994, 517-542.
12. Seidl T., Kriegel H.-P.: *Efficient User-Adaptable Similarity Search in Large Multimedia Databases.* Proc. 23rd Int. Conf. on Very Large Data Bases (VLDB), 1997, 506-515.
13. Smith J. R.: *Integrated Spatial and Feature Image Systems: Retrieval, Compression and Analysis*. Ph.D. thesis, Graduate School of Arts and Sciences, Columbia University, 1997.
14. White D.A., Jain R.: *Similarity indexing with the SS-tree.* Int. Conf on Data Engineering (ICDE), 1996.
15. Weber R., Schek H.-J., Blott S.: *A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces.* Int. Conf. on Very Large Databases (VLDB), 1998, 194-205.

A Framework for Supporting Interoperability of Data Warehouse Islands Using XML

Oscar Mangisengi, Johannes Huber, Christian Hawel, and Wolfgang Essmayr

Software Competence Center Hagenberg GmbH
Softwarepark Hagenberg, Hauptstrasse 99,
A-4232 Hagenberg, Austria
{oscar.mangisengi, johannes.huber, christian.hawel, wolfgang.essmayr}@scch.at

Abstract. Providing integrated access to islands of data warehouses has become one of the leading issues in data warehouse research and industry. This paper presents a framework for providing interoperability of distributed, heterogeneous, and autonomous data warehouses based on a federated approach. We focus on schema integration using XML, which offers significant benefits for interoperability, since it is a standardized, vendor and platform independent format for data and – in our case – metadata exchange.

1 Introduction

Data warehouses (DWHs) have become the enabling technology for analytic applications. A data warehouse is a repository of data that has been extracted, transformed, and integrated from multiple, independent data sources like operational databases or external systems [9]. The purpose of data warehouses is enterprise integration [7]. They are designed to support strategic business applications, disseminate knowledge, and enhance interactions with customers and companies.

Since DWHs are growing in size [5] enterprises increasingly end up with multiple "islands" of DWHs that are operated separately and cannot be accessed in a homogeneous way. The traditional approach to overcome this problem is to build enterprise wide DWHs, which apparently cause two problems according to [7]: (1) the enterprise warehouse is invariably overloaded and (2) building an enterprise wide solution is always an extremely time and cost consuming effort that is improbable to be successful. Therefore, the interoperability of heterogeneous, autonomous, and distributed DWHs following a federated approach has become one of the leading issues of data warehousing in research and industry, lately.

Federated environments [13] provide an adequate and promising architecture for this problem since they often reflect the real-world situation of multiple autonomous organizations having particular goals in common. The architecture of federated environments is normally divided into multiple layers (compare e.g. [15]). We follow this approach and combine it with the mediation concept proposed by [14]. In our architecture federated DWHs consist of the *local layer*, the *mediation layer*, the *federated layer*, and the *client layer*.

This paper presents a framework for supporting interoperability among arbitrary DWH islands in federated environments based on the *extensible Mark-up Language* (XML). It focuses on the federated layer and mediation layer that play crucial roles for interoperability.

Related research works in the area of federated database systems, mediation between multiple information systems, and distributed data warehousing are given in [2,3,4,7,10,13,14].

The remainder of this paper is structured as follows: Section 2 presents the project environment. Section 3 presents data warehouses in federated environments. A framework for interoperability using XML is discussed in Section 4. Finally, Section 5 concludes and provides an outlook on future research.

2 Project Environment

The work described in this paper is conducted in cooperation with the *Upper-Austrian Health Insurance Organization*, which is one of nine regional bodies of the *Austrian Social Insurance Federation.* We want to illustrate the problem using a simplified example taken from our project environment. Consider the two DWH schemas given by dimensions and fact tables as follows:

- **DWH1** is used for analysing benefits paid for medical treatment of the insurants
  Time(TimeID, Day, Month, Quarter, Year)
  Insurant(InsurantID, Name, Gender, AgeDecade, SalaryLevel)
  Benefit(BenefitID, BenefitPosition, BenefitLevel2, BenefitLevel1)
  CostofBenefit(<u>TimeID</u>, <u>InsurantID</u>, <u>BenefitID</u>, Amount)
- **DWH2** is used for analysing contributions that insurants have to pay
  Time(TimeID, Day, Month, Quarter, Year)
  Insurant(InsurantID, Name, Gender, AgeDecade, SalaryLevel)
  ContributionCategory(ContributionID, ContributionCategory, ContributionGroup)
  Contribution(<u>TimeID</u>, <u>InsurantID</u>, <u>ContributionID</u>, Amount)

Now, consider the following queries presented in natural language:

- **Query 1**: Compare the SUM of the contribution amount in the first quarter of the year 2000, for the contribution category A, for ALL insurants of the province Upper Austria and the province Vienna.
- **Query 2**: Subtract SUM of the benefit amount from SUM of the contribution amount in the province Upper Austria in the first quarter of the year 2000, for the insurance of category A, and for ALL insurants.

Since the contribution amount is physically stored in distributed DWHs query 1 could not be answered with the existing OLAP applications. For query 2, two different facts of different DWH schemas have to be compared, which could not be answered either. Consequently, a federated DWH schema is needed that allows comparing benefits and contributions within one province on the one side, as well as contributions and/or benefits among all provinces within Austria on the other side. The required environment as well as a framework based on XML is described in the following sections.

3 Data Warehouses in Federated Environments

Data warehouse federations integrate summarized data rather than copying it into local data warehouses or data marts. They aim to make a collection of independent data warehouse sources in different places appear logically to be a single data source and make information in a data warehouse available to people outside the community that primarily uses the warehouse. By integrating summarized data, users are able to access data from various data warehouse systems on various platforms.

We adopt the multi layer approach as used by [8] and extend it with additional mediation components proposed by [14] in order to augment modularity and flexibility. The resulting architecture is illustrated in Figure 1 and each layer is explained as follows:

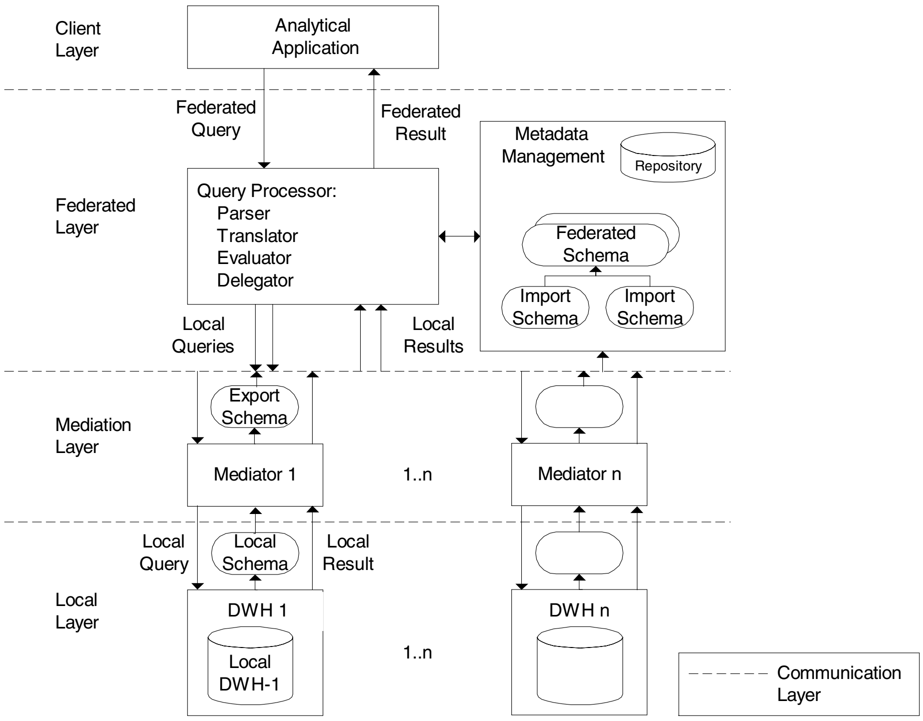


Fig. 1. A layered architecture for federated data warehouses

- **Local layer**. It consists of islands of DWHs that continue to operate with existing local applications. However, it additionally accepts local queries from the mediation layer and returns the adequate results to the respective mediator. Additionally, each DWH of the local layer is described within its local schema, which has to be provided to the mediation layer for participating within a DWH federation.

- **Mediation layer**. The mediation layer is an interface between the federated and the local layer and consists of multiple mediator modules. A mediator receives subqueries from the federated layer (i.e. the query processor) and translates them into the query language of the local DWH. It further reorganizes the data structures of the results and redistributes it to the federated layer. Additionally, the mediator translates the local schema into the *export schema*, which follows the canonical data model used to overcome heterogeneity (see Section 4.1). In order to provide autonomy the export schema only contains those parts of the local schema that the local DWH is willing to export to the DWH federation.
- **Federated layer**. The federated layer provides services to hide multiple DWHs by means of *federated schemas*. It allows for restructuring and merging of data and schemas in order to provide a homogenized view of multiple DWHs to the end users and their applications. The federated schema should be capable of expressing the multidimensionality and the hierarchy structure of dimensions according to the heterogeneity of the DWH islands. In a minimal form, the federated layer consists of a query processor and a metadata management component.
- **Client layer**. The client layer provides federated users and applications with a homogeneous interface to multiple DWHs for high-level business applications.
- **Communication layer**. It provides means for accessing other components of the architecture through a network. The communication layer is concerned with data transport, the integrity and confidentiality of the communicated data, and the authenticity of the communication peers.

4 A Framework for Interoperability Using XML

In our architecture the federated layer and the mediation layer are responsible for providing interoperability of data warehouses. The federated layer provides schema integration, and the mediation layer works as an interface between different data sources and the federated layer. In this section we focus on handling interoperability of data warehouse islands by using XML.

We use XML and XML DTD to represent schemas and data models for this framework given in Figure 2, since XML has particular advantages for interoperability. XML files are platform-independent, self-describing, and provide hierarchical information. In Figure 2 the left column shows an XML representation of schemas, such as the federated schema, import schemas, and local schemas. The right column shows the DTD documents related to the XML representation. Each DTD corresponds to a data model (i.e., canonical data model and local data models).

4.1 Canonical Data Model

Each local DWH can use its own data model. In order to be able to integrate local schemas into a federated schema, they have to be transformed into common schemas that are expressed in a common data model. The common or *canonical data model* (CDM) is a model, which is used for expressing all schema objects available in the different local schemas. [13] discusses CDM requirements for federated database systems. They considered that the CDM should be semantically rich; that is, it should provide abstraction and constraints so that the semantics relevant to schema integration can be presented. Thus, the CDM for data warehouses in federated environment allows handling design, integration, and maintenance of heterogeneous schemas of the local DWHs. It serves for describing each local schema including dimensions, dimension hierarchies, dimension levels, cubes, and measures and it should be possible to describe any schema represented by any multidimensional data model, such as star schema model, snow-flake model, and the like. For modeling the CDM we refer to [11,12].

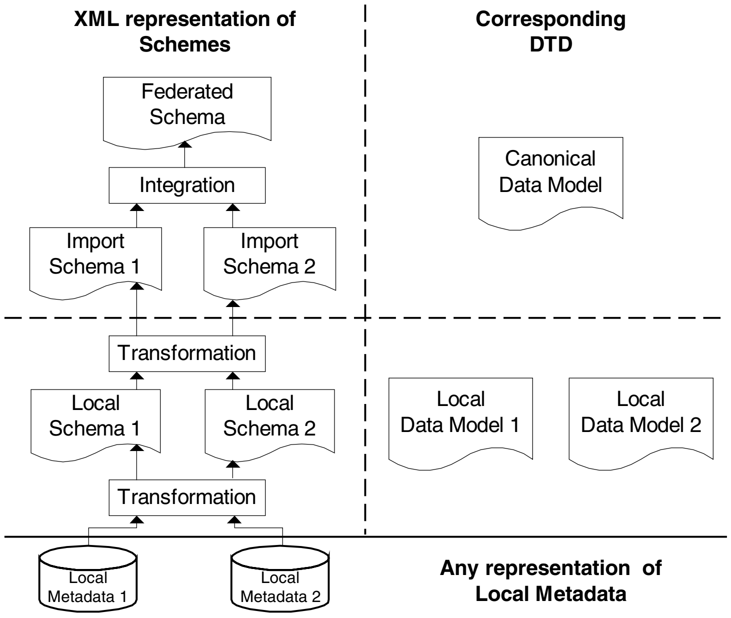


Fig. 2. XML for supporting interoperability

The CDM in Figure 3 is represented by using the *Unified Modeling Language* (UML) notation [6]. A *DWH Schema* is an extension of a relational database catalog and is a container for multidimensional storage components, namely cubes and dimensions. A dimension consists of dimension hierarchies, dimension levels, and dimension attributes, and a cube contains one or more measures. For representing the CDM we use XML DTD. XML DTD supports providing a neutral model for describing data structures. It is like the table layout of an XML document and describes the hierarchical organization. A XML DTD representation of the CDM for import schemas and federated schema looks like in Figure 4.

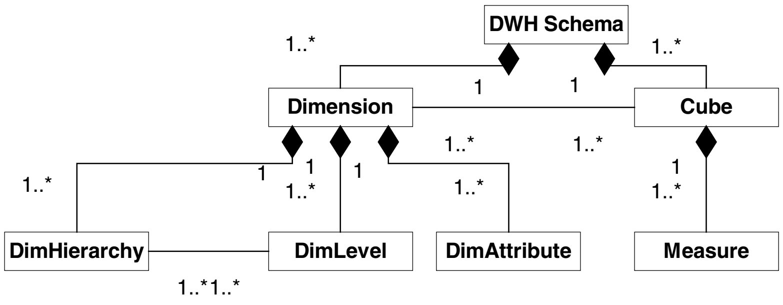


Fig. 3. The Canonical Data Model

```
<!ELEMENT DWH_Schema (Dimension+, Cube+)>
<!ATTLIST DWH_Schema SchemaName CDATA #REQUIRED>
<!ELEMENT Dimension (DimLevel+, DimAttribute*, DimHierarchy*)>
<!ATTLIST Dimension DimName ID #REQUIRED>
<!ELEMENT DimLevel (LevelAttribute+)>
<!ATTLIST DimLevel LevelID ID #REQUIRED LevelName CDATA #REQUIRED>
<!ELEMENT LevelAttribute EMPTY>
<!ATTLIST LevelAttribute LevelAttributeName CDATA #REQUIRED>
<!ELEMENT DimAttribute EMPTY>
<!ATTLIST DimAttribute DimAttributeName CDATA #REQUIRED>
<!ELEMENT DimHierarchy (LevelReference+)>
<!ATTLIST DimHierarchy HierarchyName CDATA #REQUIRED>
<!ELEMENT LevelReference EMPTY>
<!ATTLIST LevelReference
    LevelRef IDREF #REQUIRED LevelOrder CDATA #REQUIRED>
<!ELEMENT Cube (Measure+)>
<!ATTLIST Cube Name CDATA #REQUIRED DimRef IDREFS #REQUIRED>
<!ELEMENT Measure (MeasureName)>
<!ELEMENT MeasureName (#PCDATA)>
```

Fig. 4. XML DTD representation of CDM

4.2 Mediation

The mediation layer works as an interface between local DWHs and the federated DWH. The mediation layer is responsible to communicate, control, and transfer data and metadata between the local layer and the federated layer. Since mediators in our architecture are very important for interfacing heterogeneous DWHs, we consider the mediator to satisfy the following conditions, namely flexibility, locality, and modularity.

To respond to the mediation problems we suggest an approach with multiple mediators. In this approach one mediator will be attached to each local DWH. The mediator will work independently without affecting the operation of the local DWHs. In addition multiple mediators make the federated layer independent of local DWHs.

In our architecture the main responsibility of mediators is to exchange data, and to interchange metadata from the local layer to the federated layer. Thus, we use the benefits of XML, which is likely to become the standard format for interchanging metadata [12].

```
<!DOCTYPE DWH_Schema SYSTEM "CDM.dtd">
<DWH_Schema SchemaName="DWH1">
    <Dimension DimName="Time">
        <DimLevel LevelID="DimLev1" LevelName="Day">
            <LevelAttribute LevelAttributeName="Time_ID"/>
            <LevelAttribute LevelAttributeName="Day_DSC"/>
        </DimLevel>
        <DimLevel LevelID="DimLev2" LevelName="Month">
            ...
        <DimHierarchy HierarchyName="Time">
            <LevelReference LevelRef="DimLev1" LevelOrder="1"/>
            <LevelReference LevelRef="DimLev2" LevelOrder="2"/>
                ...
        </DimHierarchy>
    </Dimension>
    <Dimension DimName="Insurant">
        ...
    <Dimension DimName="Benefit">
        ...
    <Cube Name="Cost" DimRef="Time Insurant Benefit">
        <Measure>
            <MeasureName>Amount</MeasureName>
        </Measure>
    </Cube>
</DWH_Schema>
```

Fig. 5. XML document of Import Schema from DWH1

4.3 Schema Integration

This section focuses on schema integration in the federated layer for supporting the interoperability of data warehouse islands. The federated layer offers possibilities to provide interoperability among multiple, heterogeneous, distributed, and autonomous data warehouses. Our approach adheres to the tightly coupled architecture (compare [13]), saying that the federated data warehouse requires the construction of a global schema integrating information derived from source schemas. The common global schema captures the union of data available in the federation.

4.3.1 Import Schema

The export schemas from the mediation layer are called import schemas at the federated layer. An export schema represents a subset of a local schema that is available to the federated environment. Import schemas are imported from the local layer via the mediator layer using the XML format that corresponds to the canonical data model. An import schema of DWH1 corresponding to the example described in Section 2 is given in Figure 5.

```
<!DOCTYPE DWH_Schema SYSTEM "CDM.dtd">
<DWH_Schema SchemaName="FedDWH">
    <Dimension DimName="Time">
        <DimLevel LevelID="DimLev1" LevelName="Day">
            <LevelAttribute LevelAttributeName="Time_ID"/>
            <LevelAttribute LevelAttributeName="Day_DSC"/>
        </DimLevel>
        <DimLevel LevelID="DimLev2" LevelName="Month">
            …
        <DimHierarchy HierarchyName="Time">
            <LevelReference LevelRef="DimLev1" LevelOrder="1"/>
            <LevelReference LevelRef="DimLev2" LevelOrder="2"/>
                …
        </DimHierarchy>
    </Dimension>
    <Dimension DimName="Insurant">
        …
    <Dimension DimName="Benefit">
        …
    <Dimension DimName="ContributionCategory">
        ...
    <Cube Name="CostofBenefit" DimRef="Time Insurant Benefit">
        <Measure>
            <MeasureName>BenefitAmount</MeasureName>
        </Measure>
    </Cube>
    <Cube Name="Contribution" DimRef="Time Insurant ContributionCategory">
        <Measure>
            <MeasureName>ContributionAmount</MeasureName>
        </Measure>
    </Cube>
</DWH_Schema>
```

Fig. 6. XML document of Federated Schema

The first line of the XML document in Figure 5 references to an external DTD document, namely CDM.dtd for validation. The remaining lines of the XML document provide information about dimensions, dimension levels, dimension hierarchies, cubes, and measures.

4.3.2 Federated Schema

The federated schema is an integration of multiple import schemas of the local data warehouses. It is used to combine related data from multiple sites and to overcome inconsistencies in structure naming, scaling, associated behavior, and semantics. In addition, the federated schema can describe the multidimensionality of data warehouse islands, after integrating multiple import schemas.

```
<!DOCTYPE Mapping SYSTEM "Mapping.dtd">
<Mapping>
    <DimMapping FedDimName="Time">
        <MapsTo MappingType="EQ_NAMING_STRUCT">
            <ImportDimension SchemaName="DWH1" DimName="Time"/>
        </MapsTo>
        <MapsTo MappingType="EQ_NAMING_STRUCT">
            <ImportDimension SchemaName="DWH2" DimName="Time"/>
        </MapsTo>
    </DimMapping>
    <DimMapping FedDimName="Insurant">
        …
    <CubeMapping>
        <FedCube CubeName="CostofBenefit"/>
        <ImportCube SchemaName="DWH1" ImportCubeName="Cost"/>
        <MeasureMap FedMeasureName="BenefitAmount"
                    ImportMeasureName="Amount"/>
    </CubeMapping>
    <CubeMapping>
        <FedCube CubeName="Contribution"/>
            …
    </CubeMapping>
</Mapping>
```

Fig. 7. XML document of mapping information

XML and related technologies offer appropriate data management technology for documents, but also provide a neutral syntax for interoperability among disparate systems. By using XML, we can integrate schemas from local DWHs corresponding to the canonical data model, and define the structure of the document so that the validity of data can be checked. In addition, the XML document can easily be parsed into a tree, which is very useful for query processing.

The federated schema in Figure 6 shows the integration of the export schemas of DWH1 and DWH2 corresponding to the example of Section 2. The federated schema provides global *Time, Insurant, Benefit* and *ContributionCategory* dimensions and allows comparing measures of different data warehouse islands, namely *benefit amount* and *contribution amount*.

The first line of the XML document of the federated schema in Figure 6 refers to the DTD document, namely CDM.dtd for validation. The remaining part of the document shows the federated schema as a result of integrating two import schemas. The cube *CostofBenefit* depends on the dimensions *Time, Insurant and Benefit*, which is denoted by the value of the attribute *DimRef.*

4.3.3 Mapping Information

Mapping provides information about semantic heterogeneities of import schemas from local DWH schemas. Mapping information should be able to distinguish heterogeneities seen from different perspectives, namely horizontal and vertical perspectives. Horizontal perspective provides information mapping for various schemas and different applications, and vertical perspective provides mapping information for dimensions, their hierarchies, and measures. Heterogeneities of import schemas include:

- Different dimension names, but the same semantic
- The same dimension name, but different hierarchies structure
- The same measure, but functionally dependent on different dimensions

Mapping information file is created according to the XML DTD rules. The DTD file provides rules for mapping as follows:

- Dimension mapping (i.e., dimension, import dimension and its detailed mapping).
- Cube mapping (i.e., cube, import cube and its measure mapping).

Figure 7 shows the mapping between dimensions and measures of the federated schema to the import schemas. The federated dimension *Time*, denoted by the attribute *FedDimName* of element DimMapping with value “Time”, is mapped to the dimension *Time* of schema *DWH1* denoted by the attributes *SchemaName* and *DimName* of element *ImportDimension* with values “DWH1” and “Time”. The same federated *Time* dimension is mapped to the *Time* dimension of schema *DWH2*.

The federated cube *CostofBenefit* is mapped to the cube *Cost* of schema *DWH1* and the federated measure *BenefitAmount* is mapped to the corresponding measure *Amount* of the cube *Cost* of schema *DWH1*.

5 Conclusion & Future Work

In this paper we present a framework for supporting interoperability of data warehouse islands using XML. Interoperability is achieved following the federated approach [13], which allows making distributed, heterogeneous, and autonomous data sources logically appear as one single site. This virtual co-location of data warehouse islands offers advantages with respect to autonomy, compatibility, and load balancing compared to physically replicating data into specialized data marts. Concerning schema integration, the main contribution for interoperability of this framework is offered at the federated layer (i.e., canonical data model, federated schema, and import schema) and at the mediation layer (i.e. export schema).

For realizing interoperability of data warehouse islands we use the benefits of XML [1] as a standardized, universal format for data and metadata exchange. We show how different import schemas can be integrated into a federated schema using XML. Furthermore, we illustrate how XML can be used to define mapping information in order to overcome heterogeneity. However, the issue of heterogeneity has to be further investigated in order to cover all aspects of syntactic and semantic heterogeneity. XML offers an additional benefit, since all schemas can be easily parsed into a tree structure and automatically validated, which is a significant advantage for query processing.

So far, we only focused on the topic schema integration for federated data warehouses. Thus, we will address query processing for DWH federations exploring the use of XML for federated queries as well as data exchange for retrieving the results of local data warehouse islands and offering them to the federated users. Concurrently, we implement incremental prototypes demonstrating the feasibility of our approach to federated data warehouses.

References

1. Anderson, R., Birbeck, M., Kay, M., Livingstone, S., Loesgen, B., Martin, D., Mohr, S., Ozu, N., Peat, B., Pinnock, J., Stark, P., William, K.: *Professional XML.* Wrox Press Ltd., January 2000.
2. Albrecht, J., Guenzel, H., Lehner, W.: An Architecture for Distributed OLAP. *Conference Parallel and Distributed Processing Techniques and Applications* (PDPTA), Las Vegas, USA, July 13-16, 1998.
3. Albrecht. J., Lehner, W.: On-Line Analytical Processing in Distributed Data Warehouses. *International Databases Engineering and Applications Symposium* (IDEAS), Cardiff, Wales, U.K, July 8-10, 1998.
4. Bauer, A., Lehner, W.: The Cube-Query-Language (CQL) for Multidimensional Statistical and Scientific Database Systems. *Proceedings of the 5th. International Conference on Database Systems for Advanced Applications* (DASFAA), Melbourne, Australia, April 1-4, 1997.
5. Eckerson, W.W.: *Data Warehousing in the 21st. Century.* The Data Warehousing Institute, 2000. http://www.dw-institute.com/
6. Fowler, M., Scott, K.: UML Distilled A Brief Guide to the Standard Object Modeling Language. Addison Wesley, 2000.
7. Garber, L.: Michael Stonebraker on the Importance of Data Integration. IT Professional, Vol. 1, No. 3, pp. 80, 77-79, May, June 1999.
8. Gardarin, G., Finance, B., Fankhauser, P.: Federating Object-Oriented and Relational Databases: The IRO-DB Experience. *Proceedings of the 2nd. IFCIS International Conference on Cooperative Information System* (CoopIS'97), 1997.
9. Garcia-Molina, H., Labio, W., Wiener, J.L., Zhuge, Y.: Distributed and Parallel Computing Issues in Data Warehousing. *In Proceedings of ACM Principles of Distributed Computing Conference*, 1999. Invited Talk.
10. Hümmer, -W., Albrecht, J., Günzel, H.: Distributed Data Warehousing Based on CORBA. *IASTED International Conference on Applied Informatics* (AI'2000, Innsbruck, Austria, February 2000.
11. Meta Data Coalition. *Metadata Interchange Specification* (MDIS) Version 1.1, August 1997.
12. Meta Data Coalition. *Open Information Model*. Version 1.1, August 1999. http://www.mdcinfo.com/.
13. Sheth, A.P., Larson, J.A.: Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surveys*, Vol. 22, No. 3, September 1990.
14. Wiederhold, G.: Mediators in the Architecture of Future Information Systems. *The IEEE Computer Magazine*, March 1992.
15. Busse, R., Frankhauser, P., Neuhold, E.J.: Federated Schemata in ODMG. *Proceedings of the second International East-West Database Workshop*, Klagenfurt, Austria, September 1994.

Fragtique: Applying an OO Database Distribution Strategy to Data Warehouses

Marinette Savonnet and Marie-Noëlle Terrasse

Université de Bourgogne
Laboratoire d'Electronique, Informatique et Image (LE2I)
B.P. 47870 - 21078 Dijon Cedex - France
{savonnet,terrasse}@khali.u-bourgogne.fr

Abstract. We propose a strategy for distribution of a relational data warehouse organized according to a star schema. We adapt fragmentation and allocation strategies that were developed for OO databases. We split the most-often-accessed dimension table into fragments by using primary horizontal fragmentation. The derived fragmentation then divides the fact table into fragments. Other dimension tables are not fragmented since they are presumed to be sufficiently small. Allocation of fragments encompasses duplication of non-fragmented dimension tables that we call a closure.

1 Introduction

Data warehouses appeared after it became obvious that databases are not suitable for data analysis since they are designed for specific enterprise-oriented functionalities. Thus, data that are relevant for analysis are split between different bases. A data warehouse integrates data that are coming from heterogeneous sources. Those data are historical and subject-oriented. Analysis process relies on requests that group and select data. Since requests are translated into complex accesses to data warehouses, they can continue executing for hours. Thus, it is necessary to accelerate request processing, by using mechanisms such as:

References to materialized views which provide requests with pre-aggregated data, thus making request processing more efficient. The main issue, i.e., the determination of the minimal set of materialized views, has been studied by Yang & al. [16].

Indexes which make requests more efficient. Classical B+ tree-based indexes are used, as well as bitmapped or projection indexes. Datta & al. [4] have proposed to build DataIndexes that are similar to vertical partitions of the fact table.

Data distribution which enables parallelism for requests. We are interested in applying distributions that were first defined for OO structures, to new types of schemas which are used in data warehouses, such as *star schema*, *snowflake schema*, and *fact constellation* [3].

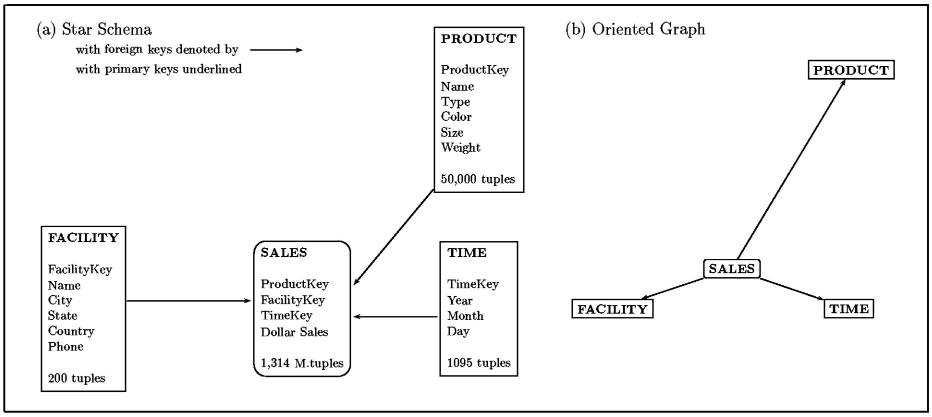


Fig. 1. Application Specification

A star schema encompasses all dimensions (including their measures) that are likely to be used by analysts. The fact table is at the center of the star: it contains all measures. Dimension tables are organized around the fact table, and all measures of the fact table share the same set of dimensions. A snowflake schema augments the star schema by providing a mechanism for expressing a hierarchy of dimensions. A snowflake schema normalizes dimension tables and formalizes a notion of hierarchy within one dimension. Similarly, fact constellation data warehouses use several fact tables in order to represent measures that are not related to the same set of dimensions.

Figure 1.a depicts a star schema. Dimension tables are not required to be in the third normal form. See, as a counter-example, the FACILITY table that contains a hierarchy (City, State, Country). The fact table is over-sized in comparison with dimension tables. Let us suppose that our data warehouse contains 200 facilities and 50,000 products, for 3 year long measures. Our FACILITY dimension table contains 200 tuples, our PRODUCT dimension table contains 50,000 tuples, and the TIME dimension table contains 1095 tuples. Thus, if the FACILITY daily-sale is 6,000 on average, the fact table SALES will contain 1,314 million tuples.

1.1 State of the art of fragmentation in data warehouses

Munneke et al. [7] propose a classification of possible fragmentation units – including horizontal, vertical and mixed fragmentation units– for a given data cube. For each type of fragment, they propose an analogy with relational fragmentation, define rules –completeness, disjointness and reconstruction of fragments– to make reconstruction of the initial data cube possible, and propose operators for fragmentation as well as for multi-dimensional reconstruction.

Specific methods apply to fragmentation of a star schema. Such schemas are characterized by a significant difference between small-sized dimension tables and a huge-sized fact table. Noaman & Barker [8] propose to derive a horizontal fragmentation of the fact table from fragmentations of certain dimension tables, as well as to duplicate non-fragmented dimension tables. Such an approach is justified by two observations. First, in most cases, some dimension tables are used more frequently than others. Second, if each dimension were fragmented, we would obtain too many fact-fragments from the fact table, i.e. as many fragments as the product of numbers of fact-fragments generated in each dimension.

However, Bellatreche [1] proposes a greedy algorithm which copes with size-differences of tables. First, a maximum number of fragments called W is chosen. Then, each step of the algorithm randomly selects a dimension table to be fragmented, and applies the derived fragmentation to the fact table. Additional steps are applied as long as cost of requests decreases and while the number of fragments remains smaller than W.

Unlike previous authors, Ezeife [5] proposes a method for fragmentation –of the fact table only– of a star schema data warehouse. All possible aggregate views are organized into a *cube lattice*, and used together with a sample of requests to describe an application. The top-level view of the cube lattice is fragmented by using the minterm algorithm for horizontal fragmentation. This processing is iterated on lower-level views which are chosen by using Harinarayanan's greedy algorithm [6]. In [15] Ezeife also proposes algorithms for updating the fragments when adding, deleting and updating data.

2 A proposal for distribution of star schema data warehouses

We propose to adapt a distribution methodology that has been developed for OO databases to distribution of star schema data warehouses. Our method, called Fragtique, relies on the assumption that –due to the OO paradigm– it is possible to determine what are the most significant data flows generated by the usual accesses to a database. We use this information to define two pre-treatment steps which aim at changing granularity of fragmentation and allocation, respectively. Our proposal is described in [2, 11].

When coping with star schema data warehouses, we assume that the structure of the data warehouse provides comparable information. Such an assumption is discussed in Section 2.1. Sections 2.2 to 2.5 describe the different steps of our method (pre-fragmentation, fragmentation, pre-allocation, allocation). Section 2.6 presents the problem of foreign keys whose values are no longer available, and a choice of solutions which can be carried out at different granularities.

2.1 From OO databases to star schema data warehouses

As we pointed out in [14], the structural point of view of an OO database relies both on the description of class-level relationships (aggregation, inheritance, etc.), and object-level relationships (relationships that appear in the bodies of methods, roles, etc.). When using OO models, those two levels are available since we have the conceptual schema, bodies of all methods, and the requirement – OO encapsulation– that every access to objects is carried out by activating a pre-defined method.

We believe that star schema data warehouses provide comparable initial information since analysis begins with dimension tables and ends with the fact table. Thus, we can determine precisely –from the star schema– which accesses to the facts are possible. In this way, the knowledge of the most often used dimensions gives us a precise model of significant data flows.

2.2 Pre-fragmentation step

We assume that the application description encompasses the star schema and a sample of requests which models accurately the activity of the application. For each request from the sample, we know its usage frequency and the tables it uses.

Pre-fragmentation defines the appropriate level of granularity for fragmentation by choosing the dimension table from which the fact table is to be partitioned. Such a dimension table is called the *principal dimension table* of the schema. In order to determine the principal dimension table, the star schema is viewed as an oriented graph in which nodes are the tables (fact and dimension tables) and edges correspond to foreign keys of the fact table. Those edges are always between the fact table and a dimension table, and they are oriented from the fact table to one of dimension tables. See Figure 1.b for the oriented graph corresponding to our example.

By studying requests, we select the principal dimension table, which also determines fragmentation of the fact table. The sub-schemas that are produced by our pre-fragmentation step are called *partitionable-trees* Such a tree either includes nodes corresponding to the fact table and its principal dimension table, or corresponds to a non-principal dimension table.

In our example, see Figure 2, the sample of requests is composed of four requests (q_1 to q_4), with requests q_1, q_2, and q_3 having higher usage frequencies. We determine three partitionable-trees: two of them (FACILITY and TIME) are non-principal dimension tables. The third tree encompasses the fact table (SALES) and the principal dimension table (PRODUCT).

Any edge between the fact table and a non-principal dimension table is called a cut-link. Cut-links can be used to evaluate activities of requests between fragments. Figure 2 shows two cut-links.

2.3 Main fragmentation step

This step creates, for each partitionable-tree, one or more distribution units called fragments. This is achieved by substituting extensions, i.e., sets of tuples, for the relations of the partitionable-tree. Classical relational fragmentation algorithms [9] are used for horizontal fragmentation of the principal dimension table. The foreign key associated with the link between the principal dimension

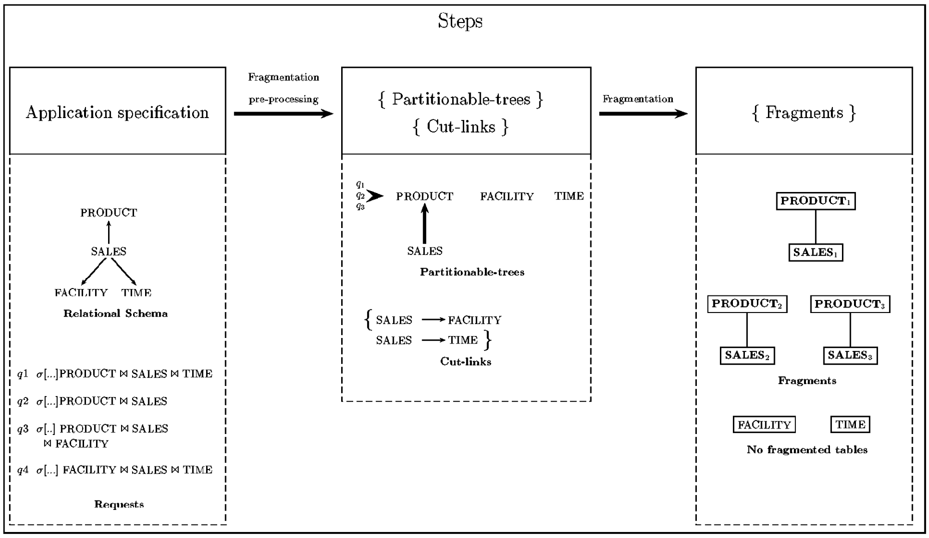


Fig. 2. Fragmentation steps in a star schema

table and the fact table is used to apply this derived fragmentation to the fact table. We obtain two types of fragments. First, we get fragments that contain both a fragment of the principal dimension table and the derived fragment of the fact table. Second, we have fragments that are limited to non-principal dimension tables that are not fragmented. The classical rules of relational database fragmentation (disjointness, reconstruction, and completeness) are satisfied by our method.

In Figure 3, we present the result of fragmentation of our earlier example which contains 20 PRODUCTS, 5 FACILITIES and a 100-days sale history. We limit our figure to display only primary keys of tables.

2.4 Pre-allocation step

Pre-allocation step aims at decreasing the complexity of calculation of the allocation cost model. For this, we group fragments into allocation-blocks such that communication costs within an allocation-block are maximized while communication costs between two allocation-blocks are minimized. Pre-allocation is based on a K-nearest-neighbours algorithm [13], in which we use, for a distance value, an request-based evaluation of communication flows between fragments.

In Figure 4.a, we present the result of evaluation of communication flows. For example, the flow between fragments $F1$ and $F4$ is evaluated to 60 based on the activities between fragments $F1$ and $F4$ that result from requests q_3 and q_4.

For each fragment, we build an ordered list (with decreasing communication flow evaluations) of fragments. We cut those lists in order to balance their sizes: we obtain three allocation-blocks ($F1$-$F4$, $F2$-$F5$, and $F3$).

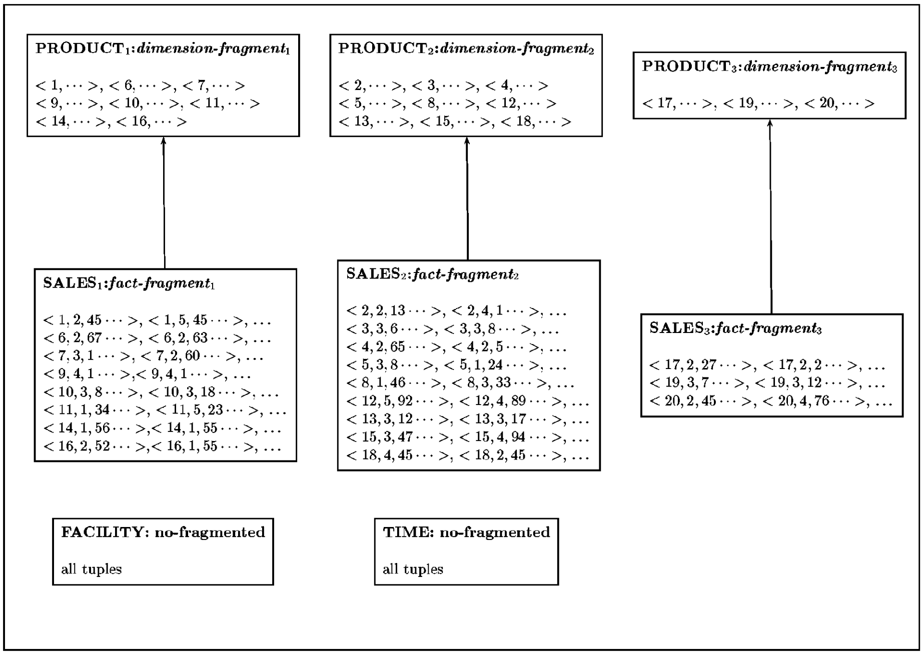


Fig. 3. Result of the fragmentation derived from PRODUCT

2.5 Allocation step

Allocation is carried out by classical cost-model-based algorithms using previous allocation-blocks as allocation units. See [12] for more details about this algorithm. The parameters we use are given in Table 1.

In order to develop our cost model, we represent a request q as a sequence of joins between the fact table (denoted by F) and dimension tables (denoted by D_i):

$$q = D_1 \bowtie F \bowtie D_2 \cdots \bowtie D_n$$

In this way, any sub-request can be described as a unique join expression.

The primary goal is to minimize an objective function that represents both storage and communication costs among distribution units. The storage cost, denoted by $StCost$, depends on the allocation schema (represented by the boolean parameters $AL_{al,S}$), the elementary storage cost and the size of the data which is stored. The storage cost is given by (see in Table 1 the meaning of parameters):

$$StCost = \sum_{S \in \{sites\}} CS_S * \sum_{al \in \{allocation-blocks\}} (AL_{al,S} * T_{al})$$

The global communication cost, denoted by $ComCost$, depends on the number of times a request q is executed on the sites. For each request q and each

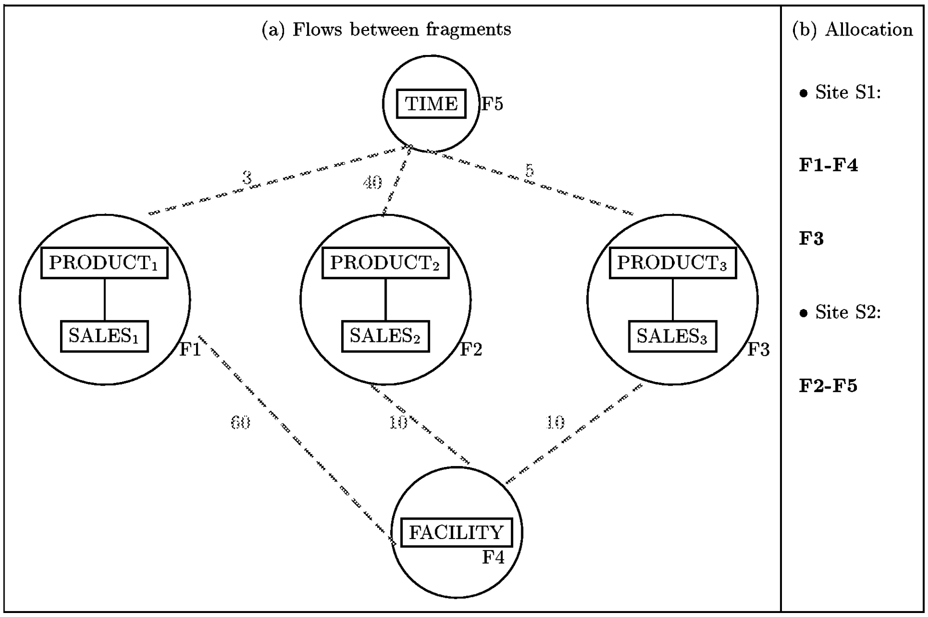


Fig. 4. Allocation steps in a star schema

site S, we assume that we know the two following parameters: $NBInv_{q,S}$, i.e., the number of times where q is executed on S, and $Inv_{q,S}$, i.e., the basic calling cost of q on S.

Thus, the global communication cost $ComCost$ is given by:

$$ComCost = \sum_{q \in \{queries\}} \sum_{S \in \{sites\}} (Inv_{q,S} * NBInv_{q,S})$$

Finally, the cost of executing a query on a site can be decomposed into two costs as follows:

1) cost of transferring results from the optimum site of execution, denoted by $CAResult_{q,S}$, which is given by:

$$CAResult_{q,S} = \sum_{al \in \{allocation-blocks\}} \min_{S' \in \{sites\}; S' \neq S} (U_{q,al} * ResultTR(q, S'))$$

Note that an execution of a query q produces sets of partial results $E(q)$ whose sizes are denoted by $Size(E(q))$. Each $E(q)$ must be transferred from its execution site S_i to the query initiating site S_0. The corresponding transfer cost is given by $ResultTR(q) = Size(E(q)) * SCost(S_i, S_0)$.

2) cost for calling sub-requests, denoted by $CASub_{q,S}$, which is evaluated as:

$$CASub_{q,S} = \sum_{q' \in \{queries\}; q' \neq q} (SubQ_{q,q'} * CAResult_{q',S})$$

sel_i	Selectivity of the predicate P_i
$\|\|t_j\|\|$	Number of tuples in T_j
$SCost(S_1, S_2)$	Elementary cost of data transfer between sites S_1 and S_2
CS_S	Cost for storage of one unit on site S
$AL_{al,S}$	Boolean-valued function; true if al is allocated on S
T_{al}	Size of allocation-block al
$NBInv_{q,S}$	Number of times q is executed on S
$U_{q,al}$	Boolean-valued function; true if q uses data in al
$SubQ_{q,q'}$	Boolean-valued function; true if the request q' is a sub-request of q

Table 1. Parameters for the cost model

Figure 4.b shows the allocation of the allocation-blocks discussed above to sites $S1$ and $S2$.

2.6 A focus on closure of sites

Our cost model relies on the assumption that when a sub-request executes on a site, all needed information is locally available. Yet in some cases, at the instance level, cut-links produce *dangling references*, i.e., foreign keys whose actual values are no longer locally available. A *closure* mechanism is proposed for suppression of dangling references, see [10] for more details. Such a closure can be carried out at different granularities: within an allocation-block, within a site, or globally. Our following examples refer to Figures 3 and 4.

Allocation-block closure corresponds to the case of dangling references whose actual values are in two different fragments of the same allocation-block. The allocation-block closure can be carried out immediately after our pre-allocation step. In our example, within the allocation-block $F1$-$F4$, $F1$ contains dangling references to FACILITY instances which belongs to $F4$.

Site closure is more general in the sense that dangling references and their actual values are required to be localized on the same site. A site closure can be carried out only after our allocation step. In our example, on the site $S1$, $F3$ contains dangling references to FACILITY instances which belongs to $F4$.

Global closure has to cope with dangling references that are not resolved by previous mechanisms. The only possible mechanism is a duplication of data in order to provide a local actual value for the dangling reference. In our example, on the site $S1$, any dangling references to TIME instances which belong to site $S2$ are to be suppressed at the global level of closure, by duplicating relevant TIME instances from $S2$ to $S1$.

3 Conclusion

We have proposed a method of distribution of star schema data warehouses that is based upon the idea of splitting the schema into autonomous and independent sub-schemas. By analyzing sample requests, we can determine which dimension tables are the most-often-used. Thus, we can choose to cut links that have lower costs in terms of data communication.

The sub-schema containing the fact table is horizontally fragmented (primary fragmentation of the dimension table and derived fragmentation of the fact table). Other sub-schemas are not fragmented.

Classical allocation algorithms are then used to allocate fragments to sites. During the whole distribution process, closure mechanisms are used to diminish dangling references by duplicating information.

In the short term, our ongoing work is to translate –as efficiently as possible– the formalization of our method into the context of data warehouses. The intention is to take into account the fact that data warehouse requests have a much simpler structure than general OO requests. As a next step, we wish to extend this proposal to snowflake and fact constellation data warehouses.

References

1. Ladjel BELLATRECHE. Une méthodologie pour la fragmentation dans les entrepôts de données. *XViiième Congrès INFORSID*, pages 246–267, May 2000.
2. Djamal BENSLIMANE, Eric LECLERCQ, Marinette SAVONNET, Marie-Noëlle TERRASSE, and Kokou YÉTONGNON. Distributing Object-Oriented Databases. *Journal of Computing Information*, 1999.
3. Surajit CHAUDHURI and Umeshwar DAYAL. An Overview of Data Warehousing and OLAP Technology. *SIGMOD Record*, 26(1):65–74, March 1997.
4. Anindya DATTA, Bongki MOON, and Helen THOMAS. A Case for Parallelism in Data Warehousing and OLAP. *9th International Workshop on Database and Expert System (DEXA98)*, pages 226–231, August 1998.
5. C.I. EZEIFE. Selecting and Materializing Horizontally Partitioned Warehouse Views. *Data & Knowledge Engineering*, 36(2):185–210, 2001.
6. V. HARINARAYANAN, A. RAJARAMAN, and J. ULLMAN. Implementing Data Cubes Efficiently. *ACM SIGMOD International Conference on Management of Data*, pages 205–216, June 1996.
7. Derek MUNNEKE, Kirsten WAHLSTROM, and Mukesh MOHANIA. Fragmentation of Multidimensional Databases. *Proceedings of the Tenth Australian Database Conference*, pages 153–164, January 1999.
8. Amin Y. NOAMAN and Ken BARKER. A Horizontal Fragmentation Algorithm for the Fact Relation in a Distributed Data Warehouse. *8th International Conference on Information and Knowledge Management (CIKM'99)*, pages 154–161, November 1999.
9. M. Tamer ÖZSU and Patrick VALDURIEZ. *Principles of Distributed Database Systems.* Prentice Hall, 1991.
10. Marinette SAVONNET, Marie-Noëlle TERRASSE, and Kokou YÉTONGNON. Using Structural Schema Information as Heuristics for the Design of Distributed Object Oriented Databases. Technical Report 96-01, Laboratoire LE2I, Université de Bourgogne, France, 1996.
11. Marinette SAVONNET, Marie-Noëlle TERRASSE, and Kokou YÉTONGNON. Fragtique: An OO Distribution Design Methodology. In A. Chen and F. Lochvsky,

editors, *Proceedings of the sixth International Conference on Database Systems for Advanced Applications (DASFAA'99)*, pages 283–290, Hsinchu – Taiwan, April 1999. IEEE Computer Society Press, ISBN=9 780769 500843.
12. Marinette SAVONNET, Marie-Noëlle TERRASSE, and Kokou YÉTONGNON. Object Clustering Methods and a Cost Model for the Design of Distributed Object-Oriented Databases. In Wu Olariu, editor, *Proceedings of the 12th ISCA International Conference on Parallel and Distributed Computing Systems (PDCS'99)*, 1-880843-29-3, pages 69–74, 1999.
13. A. SORENSEN and L. CHURCH. A comparison of strategies for data storage reduction in location-allocation problems. Technical report, NCGIA, University of California, 1995.
14. M.N. TERRASSE, M. SAVONNET, and G. BECKER. A UML-Based Metamodeling Architecture for Database Design. In *The International Database Engineering & Applications Symposium, IDEAS 2001*, July 2001. To appear.
15. Mei XU and C.I. EZEIFE. Maintaining Horizontally Partitioned Warehouse Views. *Proceedings of the Second International Conference on Data Warehousing and Knowledge Discovey (DAWAK'00)*, pages 126–133, September 2000.
16. Jian YANG, Kamalakar KARLAPALEM, and Qing LI. Algorithms for Materialized View Design in Data Warehousing Environment. *Proceedings of International Conference on Very Large Databases*, pages 136–145, August 1997.

Approximate Query Answering Using Data Warehouse Striping

Jorge Bernardino[1], Pedro Furtado[2], and Henrique Madeira[2]

[1] Institute Polytechnic of Coimbra, ISEC, DEIS, Apt. 10057
P-3030-601 Coimbra, Portugal
jorge@isec.pt

[2] University of Coimbra, DEI, Pólo II
P-3030-290 Coimbra, Portugal
{pnf, henrique}@dei.uc.pt

Abstract. This paper presents an approach to implement large data warehouses on an arbitrary number of computers, achieving very high query execution performance and scalability. The data is distributed and processed in a potentially large number of autonomous computers using our technique called data warehouse striping (DWS). The major problem of DWS technique is that it would require a very expensive cluster of computers with fault tolerant capabilities to prevent a fault in a single computer to stop the whole system. In this paper, we propose a radically different approach to deal with the problem of the unavailability of one or more computers in the cluster, allowing the use of DWS with a very large number of inexpensive computers. The proposed approach is based on approximate query answering techniques that make it possible to deliver an approximate answer to the user even when one or more computers in the cluster are not available. The evaluation presented in the paper shows both analytically and experimentally that the approximate results obtained this way have a very small error that can be negligible in most of the cases.

1 Introduction

Data warehousing refers to "a collection of decision support technologies aimed at enabling the knowledge worker (executive, manager, analyst) to make better and faster decisions" [6]. A data warehouse is a global repository that stores large amounts of data that has been extracted and integrated from heterogeneous, operational or legacy systems. OLAP is the technique of performing complex analysis over the information stored in a data warehouse [8]. The data warehouse coupled with OLAP enable business decision makers to creatively analyze and understand business trends since it transforms operational data into strategic decision making information. Typical warehouse queries are very complex and ad hoc and generally access huge volumes of data and perform many joins and aggregations. Efficient query processing is a critical requirement for data warehousing because decision support applications typically require interactive response times.

In this paper, we assume that a multidimensional database is based on a relational data warehouse in which the information is organized as a star schema [12]. A star schema is composed by a set of dimension and fact tables where the fact table accounts for most of the space occupied by all the tables in the star for most of the cases [13]. However, this table is highly normalized, which means that it really represents the most effective (concerning storage space) relational way to store the facts. The dimension tables are very denormalized, but these tables usually represent a small percentage of the space in the star. The star schema is also particularly optimized for the execution of complex queries that aggregate a large amount of data from the fact table.

We use a new round-robin data partitioning approach for relational data warehouse environments proposed in [5]. This technique, called data warehouse striping (DWS), takes advantage of the specific characteristics of star schemas and typical data warehouse queries profile to guarantee optimal load balance of query execution and assures high scalability. In DWS, the fact tables are distributed over an arbitrary number of workstations and the queries are executed in parallel by all the workstations, guaranteeing a nearly linear speedup and significantly improving query response time.

In spite of the potential dramatic speedup and scaleup that can be achieved by using the DWS technique, the fact that the data warehouse is distributed over a large number of workstations (nodes) greatly limits the practical use of the technique. The probability of having one of the workstations in the system momentarily unavailable cannot be neglected for a large number of nodes. The obvious solution of building the system using fault tolerant computers is very expensive and will contradict the initial assumption of DWS technique of using inexpensive workstations with the best cost/performance ratio. Nevertheless, high availability is required for most data warehouses, especially in areas such as e-commerce, banks, and airlines where the data warehouse is crucial to the success of the organizations.

In this paper, we propose a new approximate query answering strategy to handle the problem of temporarily unavailability of one or more computers in a large data warehouse implemented over a large number of workstations (nodes) using the DWS technique. In the proposed approach the system continues working even when a given number of nodes are unavailable. The partial results from the available nodes are used to return an estimation of the results from the unavailable nodes. Currently, we provide approximate answers for typical aggregation queries providing the user with a confidence interval about the accuracy of the estimated result. The analytic and experimental study presented in this paper show that the error introduced in the query results can be really very small.

The rest of the paper is organized as follows. In the next section, we give an overview of related work and discuss the problems associated with approximate answering in data warehouses. Section 3 briefly describes the DWS approach and section 4 discusses approximate query answering using DWS in the presence of node failures. Section 5 analyzes the experimental results and the final section contains concluding remarks and future work.

2 Related Work

Statistical techniques have been applied to databases in different tasks for more than two decades (e.g. selectivity estimation [14]). Traditionally, researchers are interested in obtaining exact answers to queries, minimizing query response time and maximizing throughput. In this work, we are interested in analyzing and giving solutions to the failure of one or more workstations in a DWS system. Thus, it has some similarities with approximate query answering research, where the main focus is to provide fast approximate answers to complex queries that can take minutes, or even hours to execute. In approximate query answering the size of the base data is minimized using samples, which is analogous to the failure of a workstation inhibiting the access to the part of the base data that resides in that workstation. A survey of various statistical techniques is given by Barbara et al [4].

Recently, there has been a significant amount of work on approximate query answering [1, 10, 16]. One of the first works was [11] where the authors proposed a framework for approximate answers of aggregation queries called online aggregation, in which the base data is scanned in random order at query time and the approximate answer is continuously updated as the scan proceeds. The Approximate Query Answering (AQUA) system [9] provides approximate answers using small, pre-computed synopses of the underlying base data. Other systems support limited on-line aggregation features; e.g., the Red Brick system supports running COUNT, AVG, and SUM [11].

Our work is also related to distributed processing in data warehouses. The fact that many data warehouses tend to be extremely large in size [6] and grow quickly means that a scalable architecture is crucial. In spite of the potential advantages of distributed data warehouses, especially when the organization has a clear distributed nature, these systems are always very complex and have a difficult global management [2]. On the other hand, the performance of distributed queries is normally poor, mainly due to load balance problems.

In this context, the DWS concept provides a flexible approach to distribution, inspired in both distributed data warehouse architecture and classical round-robin partitioning techniques. The data is partitioned in such a way that the load is uniformly distributed to all the available workstations and, at the same time, the communication requirements between workstations is kept to a minimum during the query computation phase. This paper marries the concepts of distributed processing and approximate query answering to provide a fast and reliable relational data warehouse.

3 Data Warehouse Striping

Star schemas provide intuitive ways to represent the typical multidimensional data of businesses in a relational system. In the data warehouse striping (DWS) approach, the data of each star schema is distributed over an arbitrary number of workstations having the same star schema (which is the same schema of the equivalent centralized version). The dimension tables are replicated in each machine (i.e., each dimension has exactly the same rows in all the workstations) and the fact data is distributed over the fact tables of each workstation using a strict row-by-row round-robin partitioning approach (see Figure 1). Each workstation has *1/N* of the total amount of fact rows in the star, with *N* being the number of workstations.

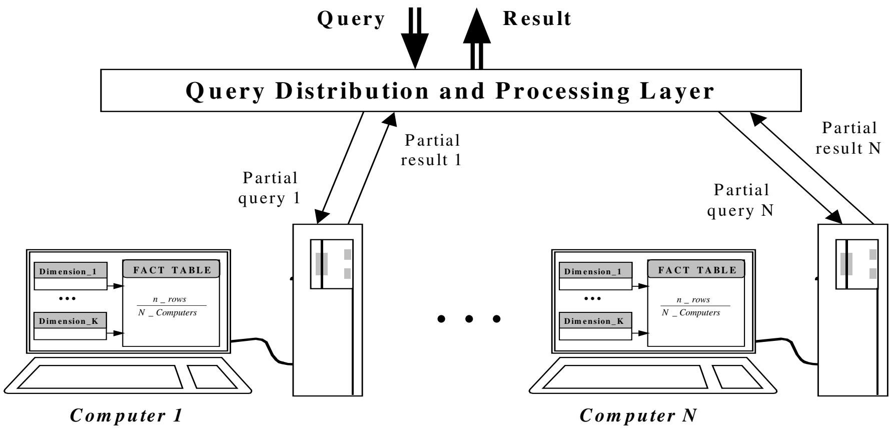


Fig. 1. Data Warehouse Striping

Most of the queries over a star schema can be transformed into *N* independent partial queries due to their nature (and because the fact rows are partitioned in a disjoint way: i.e., the rows stored in a workstation are not replicated in other workstation).

Although this is not the main point of this paper, it is important to mention that DWS achieves an optimal speedup. We have made experiments with 3, 5 and 10 workstations using a set of typical queries from APB-1 benchmark [3] and we obtained an average speedup of 3.01, 5.11 and 11.01, respectively. These results show an optimal speedup for all configurations. In fact, the speedup is higher than the theoretical value, because the centralized data warehouse that was used as the reference experiment worked near the workstation memory and I/O limits. Although the speedup increases as the number of workstations used in a DWS system increases, the probability of unavailability of a subset of those workstations also increases proportionally.

In practice, this means that DWS could not be used with a large number of workstations because the DWS will be unstable. This problem therefore would impair the use of this simple technique. However, we propose a radical solution to this problem, which allows even ordinary computers to be used as DWS computing units (without any special characteristic of hardware or software fault tolerance). In the proposed solution, the system continues running normally, even if one or more workstations are unavailable, and an approximate answer is computed from those that are available. Additionally, confidence intervals are also returned with the estimated result.

We experimentally analyzed our proposal with different configurations (different number of workstations) and in different scenarios of unavailability. The extreme case of unavailability will also be analyzed (when the user can only access his own workstation), as the approximate result obtained this way can still be useful when s/he is in the phase of “digging” the data and the precision of the result is not a crucial issue.

4 Approximate Query Answering in the Presence of Node Failures

Our proposal consists of providing the user with an answer even when one of the machines in the DWS system has a failure. In this case the answer will be approximate, because some partial results of the failed machines are unavailable. DWS is working normally with approximate answers until we manually recover the workstation. However, we show that this solution is acceptable for the following reasons:

- The error is very small, as will be shown in this paper and a small error is not relevant for the decision support activities in most of the cases;
- It is possible to provide the user with a confidence interval that gives him/her an idea of the amount of error in the approximate result.

For queries using aggregation functions an approximate answer is simply an estimated value for the answer given together with an accuracy value in the form of confidence intervals. We provide confidence intervals based on large sample bounds [10].

4.1 Estimated Values

In this section, we will see how DWS compute the approximate aggregation values when one or more workstations cannot contribute to the final result. Consider the number of workstations used in DWS to be $N = N_u + N_a$, where N_u is the number of workstations that are unavailable and N_a is the number of workstations that are available, contributing to compute the estimated aggregation value. If the aggregations functions to compute are average, sum or count and one or more workstations are unavailable the approximate average, sum and count are simply given by

$$AVERAGE_{estimated} = \frac{SUM_a}{COUNT_a}, \quad SUM_{estimated} = SUM_a \frac{N}{N_a}, \quad COUNT_{estimated} = COUNT_a \frac{N}{N_a} \tag{1}$$

where SUM_a and $COUNT_a$ represents the partial sum and count from the available workstations and N is the number of workstations used in the DWS system. Intuitively, the overall estimated average is equal to the average taken from the available nodes. These are the formulas that will be used in our experiments to compute the estimated values of the queries.

4.2 Analysis of the Error Incurred in DWS Estimations

When one or more workstations are unavailable, approximate query answers must be given. Although it is not possible to return exact answers in those cases, the estimation is extremely accurate for an important subset of typical query patterns consisting of aggregations of values into categories. The estimation is based in statistical inference using the available data as samples. We assume that the random sample is taken from an arbitrary distribution with unknown mean μ and unknown variance •^{2}. We make the additional assumption that the sample size n_s is large enough (n_s>30) so that the Central Limit Theorem can be applied and it is possible to make inferences concerning the mean of the distribution [7]. As • is unknown, we replace it with the estimate s, the sample standard deviation, since this value is close to • with high probability for large values of n_s. Thus, bounds on the confidence interval for the mean of an arbitrary distribution are given by $\overline{x}$ ±Error, where

$$\text{Error} = \quad z_{/2} \quad \frac{s}{\sqrt{n_s}}\sqrt{\frac{n \quad n_s}{n \quad 1}} \;. \qquad (2)$$

In this expression, s is the standard deviation of the sample and n is the population size. The term $z_{/2}$ is the corresponding percentile in the normal distribution. This expression shows that the error decreases significantly as the sample size n_s increases and eventually reaches extremely small values for very large sample sizes ($n_s \quad n$).

The distribution of the fact table rows into N workstations is considered pseudo-random because a round-robin approach is used. As a result, we assume the values in any aggregation pattern to be evenly distributed by all the workstations. For simplicity, we consider that an average is being computed over each aggregation group. We also consider that N_u workstations are unavailable (cannot contribute with partial values to the final query result). Some reasonable assumptions can be made concerning the values taken by these variables,

- $1 < N$ • 100
- N_u is typically a small fraction of N

Consider also an aggregation of n_g values into one of the group results, with n_g being reasonably large (e.g. n_g • 100). The number of values available in the sample when N_u workstations are unavailable is $n_g - n_g/N \quad N_u = n_g\,(1-N_u/N)$ and the error is:

$$\text{Error} = \quad z_{/2} \quad \frac{s}{\sqrt{n_g(1 \quad N_u/N)}}\sqrt{\frac{n_g \quad n_g(1 \quad N_u/N)}{n_g \quad 1}} \qquad z_{/2} \quad \frac{s}{\sqrt{n_g(1 \quad N_u/N)}}\sqrt{\frac{N_u}{N}} \qquad (3)$$

This value is typically extremely small for the type of queries considered, because the number of values aggregated into a specific group (n_g) is at least reasonably large (e.g. more than 100 values) and the fraction N_u/N is usually small. In other words, a sufficiently large sample is usually available, resulting in very accurate estimations. In these formulas we are concerning about the mean of the distribution but if we would like to compute the sum or count is only multiply the formulas above by the number of elements in each group (n_g). Figure 2 shows the 99% interval for the error taken as a function of the fraction of workstations that are unavailable (x axis) and considering also different numbers of values aggregated in a group. These results were obtained by considering the standardized normal distribution $\mathcal{N}(0,1)$.

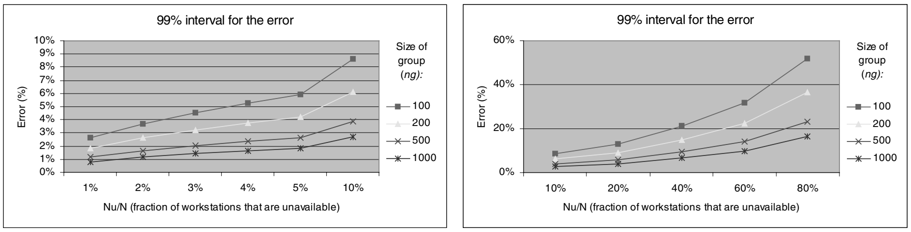


Fig. 2. 99% Confidence intervals for the error

The results of the left picture show that the error is very small when the fraction of workstations that are unavailable is reasonably small or the number of values aggregated into the group is sufficiently large. Additionally, although the increase in the fraction of workstations that are unavailable results in larger errors, as shown in the right picture, those errors are not very large in many cases and decrease significantly as n_g increases. For instance, it can be seen from the left picture that the error is extremely small when the fraction of unavailable workstations is less or equal to 10% and the number of values aggregated into the group is larger than 200.

5 Experimental Results

In these experiments we evaluate the error and test if it is within the estimated confidence intervals in a large variety of cases. We are also interested in analyzing the influence of group-by queries with different sizes of groups.

The TPC-H benchmark [15] was used as an example of a typical OLAP application. It consists of a suite of business oriented ad hoc queries illustrating decision support systems that examine large volumes of data, execute queries with a high degree of complexity and give answers to critical business questions. Therefore, in accordance with the above criteria, we concentrated our attention on queries Q1, Q6 and Q7 of TPC-H benchmark. These queries have the characteristics shown in Table 1, where the first column represents the number of rows processed in average for each group, the second column show the number of groups and the third represents the average group selectivity when the data warehouse is centralized in one workstation.

Table 1. Characteristics of queries Q1, Q6 and Q7

	number of rows/group	number of groups	selectivity
Q1	1,479,417	4	14.6 %
Q6	114,160	1	1.9 %
Q7	1,481	4	0.025 %

5.1 Experimental Testbed

The experimental evaluation of approximate query answering in DWS is conducted in a workstation environment where all are linked together in an Ethernet network with Oracle 8 database management system installed in each of them.

The TPC-H was implemented with a scale factor of 1 for the test database, which corresponds, to approximately 1 GB for the database size. This corresponds to a big fact table LINEITEM (6,001,215 rows) and the dimension tables ORDERS (1,500,000 rows), CUSTOMER (150,000 rows), SUPPLIER (10,000 rows) and NATION (25 rows).

In these experiments we apply our technique to one workstation, simulating a centralized data warehouse and denote it as CDW (Centralized Data Warehouse), and to N=5, 10, 20, 50 and 100 workstations, which corresponds to DWS-5, DWS-10, DWS-20, DWS-50 and DWS-100, respectively.

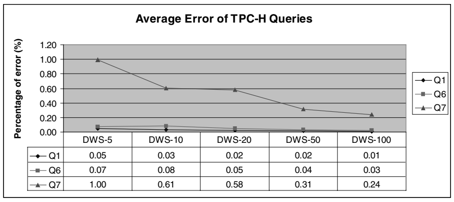
The use of *N* workstations was simulated by dividing the *n_fact_rows* (6,001,215) of LINEITEM fact table, into *N* partial fact tables (LINEITEM_1,…, LINEITEM_N). Each workstation has *n_fact_rows/N* rows and the dimensions are replicated in each workstation. For example, DWS-100 simulates the use of 100 workstations (*N*=100) having 100 partial fact tables (LINEITEM_1, … ,LINEITEM_100) with each one having 600,121 1 fact rows, while the dimensions are equivalent to those of the CDW system.

5.2 Approximate Query Answers

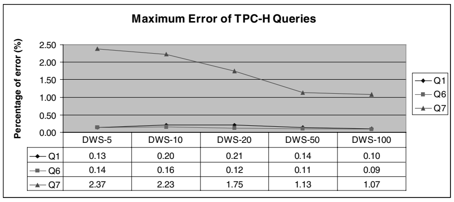
In these experiments we evaluated the error obtained with typical OLAP queries when some of the workstations are unavailable and proved that we can give very tight confidence intervals such that users know about the accuracy of the result. The influence of group-by queries with different sizes of groups will also be analyzed.

Estimation accuracy. The unavailability of each individual workstation was simulated by not taking into account the partial results that corresponded to the failed workstation. Finally, we compute the average and maximum of all these errors for each configuration. For example, using the DWS-100 configuration, we determine the error when one of the 100 workstations is unavailable and determine the average and maximum of these errors.

The average and maximum error obtained for Q1, Q6 and Q7 queries of TPC-H benchmark using DWS-5, DWS-10, DWS-20, DWS-50 and DWS-100 and considering only one unavailable workstation are illustrated in Figure 3, where the *x* axis represents the number of workstations used.



	DWS-5	DWS-10	DWS-20	DWS-50	DWS-100
◆ Q1	0.05	0.03	0.02	0.02	0.01
■ Q6	0.07	0.08	0.05	0.04	0.03
▲ Q7	1.00	0.61	0.58	0.31	0.24



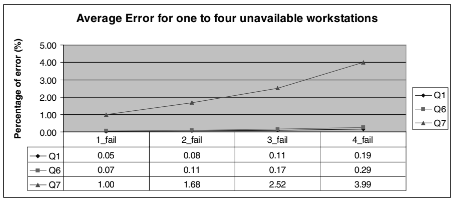
	DWS-5	DWS-10	DWS-20	DWS-50	DWS-100
◆ Q1	0.13	0.20	0.21	0.14	0.10
■ Q6	0.14	0.16	0.12	0.11	0.09
▲ Q7	2.37	2.23	1.75	1.13	1.07

Fig. 3. Average and maximum error for Q1, Q6 and Q7 when one workstation is unavailable

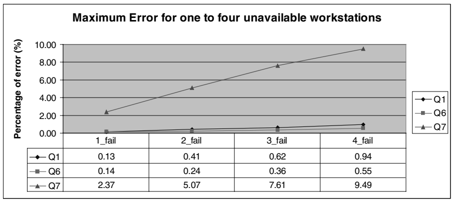
As can be observed, the average error obtained for these queries is extremely small because we only simulate the unavailability of one of the workstations that comprise each configuration. Additionally, this error decreases when we use more workstations, due to the fact that the number of missing rows is smaller. The average error obtained for Q7 query is larger than the error corresponding to the other queries because the average number of rows aggregated by group is smaller. This is due to the fact the query Q7 has a higher selectivity (as shown in Table 1), meaning less elements in each aggregation group, which in case of failure of one workstation has more impact in the precision of the result obtained.

The maximum error is higher than average error because it is the worst-case. We compute the maximum error obtained for each query and for each group. However, as illustrated in the right picture of Figure 3, this error was always smaller than 2.5% even when 1/5 of workstations were unavailable.

In the results shown before only one workstation was unavailable, but we are also interested in studying the results when the number of unavailable workstations is much larger. For instance, in DWS-5 we can simulate the failure of 2, 3 or 4 workstations, which corresponds to an unavailability of 40%, 60% or 80%, respectively. The average and maximum error for all possible combinations is shown in Figure 4 for queries Q1, Q6 and Q7, where the *x* axis represents the number of workstations unavailable.



	1_fail	2_fail	3_fail	4_fail
Q1	0.05	0.08	0.11	0.19
Q6	0.07	0.11	0.17	0.29
Q7	1.00	1.68	2.52	3.99



	1_fail	2_fail	3_fail	4_fail
Q1	0.13	0.41	0.62	0.94
Q6	0.14	0.24	0.36	0.55
Q7	2.37	5.07	7.61	9.49

Fig. 4. Average and maximum error on DWS-5 when one to four workstations are unavailable

In these experiments, using DWS-5 configuration, the error increases with the number of workstations that are unavailable (as expected). However, this error is not very large in average, as it does not exceed 4% (Q7 query) or even less. Furthermore, the average error is not higher than 0.3% for queries Q1 and Q6, which is a very good approximation of the result for most of the cases. These very good results are due to the fact that our partitioning strategy is pseudo-random resulting in a uniform random distribution of the data over the workstations.

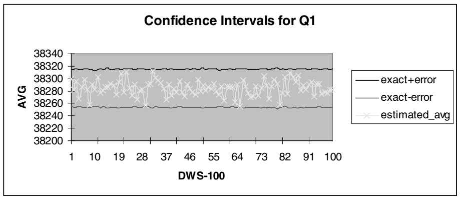
The maximum error obtained using all the possible combinations of workstation unavailability is about 10%. It must be pointed out this is the extreme case of unavailability because the user is only accessing his own workstation. Interestingly, the maximum error of Q1 query is higher than the maximum error of Q6 even though Q1 aggregates more rows in average than Q6 (see Table 1). However, this is due to the influence of a very small group in Q1. This group has 38,854 rows, which is a much smaller number of rows than those from query Q6 (see Table 1). Therefore, we could conclude that the precision of our results is highly influenced by the number of rows processed in each group of a group-by query. However, even in the case of unavailability of 80% of the workstations we obtain an error less than 10% in the worst case meaning that approximate results are not harmful.

Confidence intervals. We provide accuracy measures as confidence intervals for each estimate, for some confidence probability.

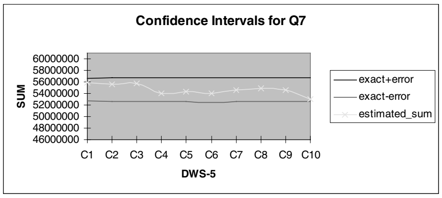
The next figures analyze the confidence interval that is returned using our technique for queries Q1 and Q7 using various configurations of DWS. Each graphic shows three curves, two of them representing the sum of the exact value with the confidence interval (*exact_value+error* and *exact_value-error*), corresponding to the upper and lower curves in the figures. The middle curve is the *estimated_value* obtained with the respective configuration.

Figure 5(a) shows the confidence interval for query Q1 using DWS-100 and the aggregation *avg(l_extendedprice)* for one of the groups of the result. As we are simulating the unavailability of only one workstation, the *x* axis legend indicates which workstation is unavailable and the *y* axis shows the value of the aggregation as well as the confidence interval bounds. This example shows that the confidence intervals do a good job determining boundaries for the error. These intervals are computed using the formula 3 of section 4.2, with a probability of 99%.

Figure 5(b) shows the confidence intervals for query Q7 and all possible combinations of unavailability of three workstations using DWS-5. The value computed by query Q7 is the aggregation *sum(volume).* The *x* axis represents all possible combinations of unavailability of three workstations. The query Q7 returns four groups, but for simplicity is only shown the result of one in the figure 5(b). In this case confidence intervals are computed with a probability of 95%.



(a) Unavailability of one workstation



(b) Combinations of unavailability of three workstations

Fig. 5. Confidence interval for queries Q1 and Q7 using DWS-100 and DWS-5

The experimental results show that confidence intervals are very useful to deliver convenient error bounds for a given confidence level and the errors are very small. Thus, we can give the user very tight confidence intervals of the approximate answers when one or more workstations are unavailable. The artificial nature of the TPC-H benchmark data could influence the results but we argue that this highly-accurate answers are mainly due to our round robin data partitioning which provides randomness of our facts which would not be the case if we have used range partitioning.

6 Conclusions and Future Work

In this paper, we have proposed a distributed data warehousing strategy that is able to answer typical OLAP queries when component workstations are unavailable. Data Warehouse Striping (DWS) is a scalable technique that divides data warehouse facts into a number of workstations to solve data warehouse limitations related to heavy storage loads and performance problems. With the proposed modular approach, simple workstations without any special hardware or software fault tolerance can be used and very accurate approximate answers are returned even when a substantial number of the component workstations are unavailable. We have proposed a formula to quantify estimation error of the answer and proved that this error is very small when the fraction of workstations that are unavailable is reasonably small or the number of values aggregated into the groups is sufficiently large.

The proposed technique is a cost-effective solution that could be applied in almost all types of organizations, taking advantage of the availability of computer networks to distribute the data and their processing power, avoiding the need of very expensive servers. The experimental results show a linear or even super linear speedup of DWS, due to the fact that, when we distribute the data, we are working with more manageable amounts of data that do not stress memory and computing resources so much.

The experimental results of this paper have also shown that the DWS technique provides approximate query answers with very small errors, even when most of the workstations are unavailable. The confidence intervals are promising, as the technique is able to return strict confidence intervals with important information to the user concerning the amount of error of the estimations. We propose a more complex statistical analysis as future work.

References

1. Acharaya, S., Gibbons, P.., Poosala, V.: Congressional Samples for Approximate Answering of Group-By Queries. ACM SIGMOD Int. Conf on Management of Data, (2000) 487-498
2. Albrecht, J., Gunzel, H., Lehner, W.: An Architecture for Distributed OLAP. Int. Conference on Parallel and Distributed Processing Techniques and Applications PDPTA, (1998)
3. APB-1 Benchmark, Olap Council, November 1998, www.olpacouncil.org
4. Barbara, D., et al.: The New Jersey data reduction report. Bulletin of the Technical Committee on Data Engineering, 20(4) (1997) 3-45
5. Bernardino, J., Madeira, H.: A New Technique to Speedup Queries in Data Warehousing. In Proc. of Chalenges ADBIS-DASFAA, Prague (2000) 21-32
6. Chauduri, S., Dayal, U.: An overview of data warehousing and OLAP technology. SIGMOD Record, 26(1), (1997) 65-74
7. Cochran, William G.: Sampling Techniques, 3rd edn, John Wiley & Sons, New York, 1977.
8. Codd, E.F., Codd, S.B., Salley, C.T.: Providing OLAP (on-line analitycal processing) to user- analysts: An IT mandate. Technical report, E.F. Codd & Associates (1993)
9. Gibbons, P. B., Matias Y.: New sampling-based summary statistics for improving approximate query answers. ACM SIGMOD Int. Conf. on Management of Data (1998) 331–342
10. Haas, P. J.: Large-sample and deterministic confidence intervals for online aggregation. In Proc. 9th Int. Conference on Scientific and Statistical Database Management (1997) 51-62
11. Hellerstein, J.M., Haas, P.J., Wang, H.J.: Online aggregation. ACM SIGMOD Int. Conference on Management of Data (1997) 171-182
12. Kimball, Ralph: The Data Warehouse Toolkit. Ed. J. Wiley & Sons, Inc (1996)
13. Kimball, Ralph, Reeves, L., Ross, M., Thornthwalte, W.: The Data Warehouse Lifecycle Toolkit. Ed. J. Wiley & Sons, Inc (1998)
14. Selinger, P., et al.: Access Path Selection in a Relational Database Management System. ACM SIGMOD Int. Conf. on Management of Data (1979) 23-34
15. TPC Benchmark H, Transaction Processing Council, June 1999, www.tpc.org
16. Vitter, J., Wang, M.: Approximate computation of multidimensional aggregates of sparse data using wavelets. ACM SIGMOD Int. Conf. on Management of Data (1999) 193-204

Author Index