Bart Kuijpers
Peter Revesz (Eds.)

LNCS 3074

Constraint Databases

First International Symposium, CDB 2004
Paris, France, June 2004
Proceedings

Springer

Lecture Notes in Computer Science 3074

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

Springer
Berlin
Heidelberg
New York
Hong Kong
London
Milan
Paris
Tokyo

Bart Kuijpers Peter Revesz (Eds.)

Constraint Databases

First International Symposium, CDB 2004
Paris, France, June 12-13, 2004
Proceedings

Springer

eBook ISBN: 3-540-25954-6
Print ISBN: 3-540-22126-3

Visit Springer's eBookstore at: http://ebooks.springerlink.com
and the Springer Global Website Online at: http://www.springeronline.com

Preface

The first International Symposium on the Applications of Constraint Databases (CDB2004) took place in Paris, France, on June 12–13, 2004, just before the ACM SIGMOD and PODS conferences.

Since the publication of the paper "Constraint Query Languages" by Kanellakis, Kuper and Revesz in 1990, the last decade has seen a growing interest in constraint database theory, query evaluation, and applications, reflected in a variety of conferences, journals, and books. Constraint databases have proven to be extremely flexible and adoptable in environments that relational database systems cannot serve well, such as geographic information systems and bioinformatics.

This symposium brought together people from several diverse areas all contributing to the practice and the application of constraint databases. It was a continuation and extension of previous workshops held in Friedrichshafen, Germany (1995), Cambridge, USA (1996), Delphi, Greece (1997), and Seattle, USA (1998) as well as of the work in the comprehensive volume "Constraint Databases" edited by G. Kuper, L. Libkin and J. Paredaens (2000) and the textbook "Introduction to Constraint Databases" by P. Revesz (2002).

The aim of the symposium was to open new and future directions in constraint database research; to address constraints over domains other than the reals; to contribute to a better implementation of constraint database systems, in particular of query evaluation; to address efficient quantifier elimination; and to describe applications of constraint databases.

The technical program of the symposium consisted of 10 technical papers and an invited paper as well as additional invited talks by Leonid Libkin and Andreas Podelski. The papers collected in these proceedings were selected by the program committee from a total of 29 submissions, and they were presented in five sessions, as described below.

Efficient query evaluation. Joos Heintz (invited speaker) and Bart Kuijpers address the difficulty of the effective evaluation of first-order queries, usually involving some form of quantifier elimination, and discuss various aspects that influence the efficiency of the evaluation of queries expressible in first-order logic over the reals. The importance of data structures and their effect on the complexity of quantifier-elimination is emphasized and a novel data model that supports data exploration and visualization as well as efficient query evaluation is proposed. Finally, they show that a particular kind of sample point query cannot be evaluated in polynomial time.

Spatial and spatio-temporal data. Spatial databases is a common application area of constraint databases. In recent years spatio-temporal data have often been modeled using constraints. We have three technical papers on this topic.

- Lixin Li, Youming Li and Reinhard Piltner propose a new spatio-temporal interpolation method for 3-D space and 1-D time geographic data, based on shape functions. Instead of only manipulating the time dimension as in the earlier ST product and tetrahedral methods, their new method takes the original approach of combining 2-D shape functions in the (x, y) domain with the (z, t) domain shape functions.
- Floris Geerts deals with the representation of moving objects in databases. Moving objects are usually represented, when possible, through explicit descriptions of their trajectories. The author proposes instead a new data model based on encoding their equations of motion, more specifically by differential equations. He also discusses a query language for this data model.
- Sofie Haesevoets describes a triangle-based logic in which queries that are invariant under affinities of the ambient space can be formulated. She characterizes the expressive power of this logic and shows it to be equivalent to the affine-generic fragment of first-order logic over the reals. She also presents algorithms for computing an affine-invariant triangulation and covering.

Applications. Looking at specific applications is important for two reasons. First, they reveal the possibilities of constraint database applications, often applications that could not be done in relational database systems. Second, they test the limits of the current constraint data model and query language proposals and thereby stimulate their further extensions. The following specific applications raise important issues and provide big challenges to researchers for the future.

- Maria Teresa Gómez López, Rafale Ceballos Guerrero, Rafael Martínez Gasca and Carmelo del Valle Sevilla apply constraint databases in the determination of potential minimal conflicts, which can be further used for polynomial model-based diagnosis.
- Viswanathan Ramanathan and Peter Revesz apply constraint databases to the genome map assembly problem. The genome map assembly problem is the problem of reconstructing the entire genome sequence of an organism based on overlapping fragments of the genome. They look at several algorithms for this problem. Using extensive computer experiments, they show that their constraint automaton, which can be solved using a constraint database system, solves the genome map assembly problem computationally more efficiently than the common alternative solution based on overlap multigraphs. Even more surprisingly, the average case running time of their solution increases only linearly while the running time of the other solution increases exponentially with the size of real genome data input.
- Carson Kai-Sang Leung proposes a new dynamic FP-Tree mining algorithm to mine frequent itemsets satisfying succinct constraints. The proposed algorithm is dynamic, such that the constraints can be changed during the mining process. Based on a classification of constraints this paper describes the cases of relaxing and tightening constraints and evaluation results show the effectiveness of this approach.

Query optimization. Query optimization is the concern of making the evaluation of queries computationally efficient in space and time. These techniques are essential elements for the implementation of constraint database systems. We had two papers in this area.

- Jan Chomicki discusses the problem of semantic query optimization for preference queries and treats this problem as a constraint reasoning problem. His techniques make use of integrity constraints, and make it possible to remove redundant occurrences of the winnow operator resulting in a more efficient algorithm for the computation of winnow. The paper also investigates the problem of propagating integrity constraints.
- Anagh Lal and Berthe Y. Choueiry consider the important problem of efficient join computation during query evaluation. They model the join computation in relational databases as a constraint satisfaction problem, which they solve using their technique called dynamic bundling. With dynamic bundling the join computation can be performed with major savings in space and time.

The future of constraint databases. Implementation of constraint databases is, of course, a major practical concern. While there are several prototype systems developed at universities and research laboratories, such as the C^3, the DEDALE and the MLPQ systems, there are still no commercial implementations of constraint databases. However, this situation may change in the future, as explained in the following two papers.

- Dina Goldin describes how constraints can be eliminated from constraint databases, in the sense of reducing them to as simple a representation as used in relational database systems and geographic information systems. She proposes a 3-tier architecture for constraint databases, with an abstract layer for the infinite relational extent of the data and a concrete layer that admits both constraint-based and geometry-based representations of spatio-temporal data.
- Mengchui Cai, from the DB2 group at the IBM Silicon Valley Laboratory, presents a way of integrating constraint databases into relational database systems. His main insight is that existing relational database systems can be extended by special functions that call a constraint relational engine at the appropriate places within an extended SQL query, while the constraint data itself can be represented within specialized relational tables. This proposal may lead to a practical and seemless way of integrating constraint data with relational data.

This symposium would have been impossible without the help and effort of many people. The editors would like to thank the program committee for the selection of the papers and the local organizers, in particular Irène Guessarian, for the arrangements in Paris. We especially would like to thank Sofie Haesevoets for managing the conference Web site and many other practical arrangements, and Floris Geerts for advertising the symposium and composing these proceedings.

The organizers are extremely grateful for the financial support given by General Eleftherios and Argyroula Kanellakis, the University of Limburg (LUC) and the University of Nebraska-Lincoln.

We would explicitly like to thank the Université Pierre et Marie Curie, Paris 6, for hosting the symposium.

We are pleased to bring to the reader these symposium proceedings, which reflect major recent advances in the field of constraint databases. We were also glad to see the symposium bring together many researchers in the field of constraint databases for a fruitful exchange of ideas. We also remembered those who due to their untimely death could not attend the symposium, including Paris Kanellakis and his family. Finally, we look forward to a continued growth in the field and to future symposium events.

June 2004 Bart Kuijpers and Peter Revesz

Conference Organization

Chairs

Bart Kuijpers	Limburgs Universitair Centrum, Belgium
Peter Revesz	University of Nebraska-Lincoln, USA

Program Committee

Saugata Basu	Georgia Tech, USA
Alex Brodsky	George Mason University, USA
Jan Chomicki	SUNY at Buffalo, USA
Berthe Choueiry	University of Nebraska-Lincoln, USA
Giorgio Delzanno	Università di Genova, Italy
Floris Geerts	University of Helsinki, Finland
Marc Giusti	CNRS, Ecole Polytechnique, Paris, France
Dina Goldin	University of Connecticut, USA
Stephane Grumbach	INRIA, Paris, France
Joxan Jaffar	National University of Singapore, Singapore
Manolis Koubarakis	Technical University of Crete, Greece
Stephan Kreutzer	Humboldt Universität, Berlin, Germany
Bart Kuijpers	Limburgs Universitair Centrum, Belgium
Gabriel Kuper	Università di Trento, Italy
Zoe Lacroix	Arizona State University, USA
Lixin Li	Georgia Southern University, USA
Jan Paredaens	Universiteit Antwerpen, Belgium
Peter Revesz	University of Nebraska-Lincoln, USA
Philippe Rigaux	Université Paris Sud, France
Kai-Uwe Sattler	Technische Universität Ilmenau, Germany
Jianwen Su	University of California at Santa Barbara, USA
David Toman	University of Waterloo, Canada
Jan Van den Bussche	Limburgs Universitair Centrum, Belgium
Dirk Van Gucht	Indiana University, USA
Nicolai Vorobjov	University of Bath, UK
Mark Wallace	Imperial College, London, UK

External Reviewers

Xiang Fu, Cagdas Gerede, Sofie Haesevoets, Anagh Lal, Hoda Mokhtar

Publicity and Proceedings Chair

Floris Geerts	University of Helsinki, Finland

This page intentionally left blank

Table of Contents

Efficient Query Evaluation

Constraint Databases, Data Structures and Efficient Query Evaluation 1
Joos Heintz and Bart Kuijpers

Spatial and Spatio-Temporal Data

A New Shape Function Based Spatiotemporal Interpolation Method 25
Lixin Li, Youming Li, and Reinhard Piltner

Moving Objects and Their Equations of Motion 40
Floris Geerts

A Triangle-Based Logic for Affine-Invariant Querying
of Two-Dimensional Spatial Data 52
Sofie Haesevoets

Applications

Applying Constraint Databases in the Determination
of Potential Minimal Conflicts to Polynomial Model-Based Diagnosis 74
Maria Teresa Gómez López, Rafael Ceballos Guerrero, Rafael Martínez Gasca, and Carmelo del Valle Sevilla

Constraint Database Solutions to the Genome Map Assembly Problem ... 88
Viswanathan Ramanathan and Peter Revesz

Dynamic FP-Tree Based Mining
of Frequent Patterns Satisfying Succinct Constraints 112
Carson Kai-Sang Leung

Query Optimization

Semantic Optimization of Preference Queries 128
Jan Chomicki

Constraint Processing Techniques for Improving Join Computation:
A Proof of Concept 143
Anagh Lal and Berthe Y. Choueiry

The Future of Constraint Databases

Taking Constraints out of Constraint Databases 161
Dina Q. Goldin

Integrating Constraint and Relational Database Systems 173
Mengchu Cai

Author Index 181

Constraint Databases, Data Structures and Efficient Query Evaluation*

Joos Heintz[1,2,3] and Bart Kuijpers[4]

[1] Universitad de Buenos Aires
Departamento de Computación
Ciudad Universitaria, Pabellón I, 1428 Buenos Aires, Argentina
joos@dc.uba.ar
[2] Consejo Nacional de Investigaciones
Científicas y Tecnológicas (CONICET), Argentina
[3] Universidad de Cantabria
Facultad de Ciencias
Departamento de Matemáticas, Estadística y Comptación
Avda. de los Castros s/n, E-39005 Santander, Spain
[4] Limburgs Universitair Centrum
Department WNI
Universitaire Campus, 3590 Diepenbeek, Belgium
bart.kuijpers@luc.ac.be

Abstract. Constraint databases that can be described by boolean combinations of polynomial inequalities over the reals have received ample research attention. In particular, the expressive power of first-order logic over the reals, as a constraint database query language, has been studied extensively. The difficulty of the effective evaluation of first-order queries, usually involving some form of quantifier elimination, has been largely neglected.

The contribution of this paper is a discussion of various aspects that influence the efficiency of the evaluation of queries expressible in first-order logic over the reals. We emphasize the importance of *data structures* and their effect on the complexity of quantifier-elimination. We also propose a novel data model that supports data exploration and visualization as well as efficient query evaluation. In this context, we introduce the concept of *sample point query*. Finally, we show that a particular kind of sample point query cannot be evaluated in polynomial sequential time by means of branching-parsimonious procedures.

1 Introduction and Summary

The framework of *constraint databases* was introduced in 1990 by Kanellakis, Kuper and Revesz [26] as an extension of the relational model that allows the

* Research partially supported by the following Argentinian, Belgian, German and Spanish grants: UBACyT X198, PIP CONICET 2461, FW/PA/02–EIII/007, ALA 01–E3/02 and DGCyT BFM 2000–0349.

B. Kuijpers and P. Revesz (Eds.): CDB 2004, LNCS 3074, pp. 1–24, 2004.

use of possibly infinite, but first-order definable relations rather than just finite relations. It provides an elegant and powerful model for applications that deal with infinite sets of points in some real affine space $\mathbb{R}^n$. In the setting of the constraint model, infinite relations are finitely represented as boolean combinations of polynomial equalities and inequalities, which we interpret, in this paper, over the real and exceptionally over the complex numbers. The case of the interpretation over the real numbers has applications in spatial databases [31].

Various aspects of the constraint model are well-studied by now (for an overview of research results we refer to [28] and the textbook [33]). The relational calculus augmented with polynomial constraints, or equivalently, first-order logic over the reals augmented with relation predicates to address the database relations $R_1, \ldots, R_s$, $\mathsf{FO}(+, \times, <, 0, 1, R_1, \ldots, R_s)$ for short, is the standard first-order query language for constraint databases. The expressive power of first-order logic over the reals, as a constraint database query language, has been studied extensively. However, the difficulty of the efficient evaluation of first-order queries, usually involving some form of quantifier elimination, has been largely neglected. The existing constraint database systems are based on general purpose quantifier-elimination algorithms and are, in most cases, restricted to work with linear data, i.e., they use first-order logic over the reals without multiplication [28, Part IV].

The intrinsic inefficiency of quantifier elimination represents a bottle-neck for real-world implementations of constraint database systems. General purpose elimination algorithms (such as, e.g., [6,12,18, 23,32]) and standard data structures prevent query evaluation to become efficient. The fact that the knapsack problem can be formulated in this setting indicates that geometric elimination may be intrinsically hard. Another example for this complexity phenomenon is given by the permanent of a generic $n \times n$ matrix, which appears as the output of a suitable first-order query (see [22] for details on both examples).

In the literature of constraint databases, the data model proposed to describe geometric figures in $\mathbb{R}^n$ is based on quantifier-free first-order formulas over the reals [28,33]. The data structures needed to implement this data model are left largely unspecified. It is widely understood that these formulas should be represented by explicitly giving disjunctive normal forms using dense or sparse encoding of polynomials. However, disjunctive normal forms may be unnecessarily large and the sparse representation of elimination polynomials may be very inefficient. For example, the sparse representation of the determinant of a generic $n \times n$ matrix contains $n!$ terms. On the other hand, the determinant can be represented by an $O(n^3)$ arithmetic boolean circuit (with divisions) which describes the Gaussian elimination algorithm. This suggests the use of alternative data structures for the representation of the classical data model of constraint databases. Indeed, the use of *arithmetic boolean circuits* as alternative data structure allows the design of a new generation of elimination algorithms which produce an exponential time complexity gain compared to the most efficient algorithms using traditional data structures (see [36] for a definition of the notion of arithmetic boolean circuit and [22] for this kind of complexity results). Nevertheless,

in terms of the syntactical size of the input, the worst-case sequential time complexity of the elimination of a single existential quantifier block by means of the new algorithms remains still exponential. However, when we measure the input in a semantic (i.e., geometric) way, as is achieved, e.g., by the *system degree,* elimination of a single existential quantifier block becomes polynomial in this quantity (see e.g. [3,4,13,15]). Unfortunately, this does not suffice for the design of algorithms able to evaluate purely existential queries in sequential time which is polynomial in the number of bounded variables. In fact, an non-polynomial lower bound for the underlying elimination problem can be deduced from the $\mathrm{P}_{\mathbb{R}} \neq \mathrm{NP}_{\mathbb{R}}$ conjecture in the Blum-Shub-Smale complexity model over the real numbers [8].

Another shortcoming of the classical data model for constraint databases is that it does not support data exploration and local visualization. Indeed, a quantifier-free formula in disjunctive normal form, describing the output of a query, allows the answering of, for instance, the membership question, but it is does not allow an easy exhibition of the output, by, e.g., the production of *sample points,* or, for low dimensions, a visualization of the output. To increase the tangibility of the output, we suggest considering a new type of query that produces sample points. Furthermore, it could be desirable to support an exploration of the neighborhood of such a sample point. This could be achieved by representing the output by a cell decomposition consisting of cells which are non-empty open subsets of smooth real varieties. In this way, starting from any sample point, its neighborhood within its cell may be explored. In this sense, we propose a novel data model for constraint databases, consisting of *smooth cells accompanied by sample points.* The known most efficient elimination procedures produce naturally such output representations, a typical example being CAD [12]. Therefore, when constraint database theory invokes quantifier elimination in query evaluation, it should also incorporate these features of the existing elimination algorithms.

In this context, we extend the concept of sample point query to queries that give rationally parameterized families of polynomial functions as output. Such queries will be called *extended sample point queries.* The main outcome of the paper is a proof that extended sample point queries, associated to first-order formulas containing a *fixed* number of quantifier alternations, cannot be evaluated in polynomial sequential time by so-called "branching-parsimonious algorithms". This lower bound result suggest that further research on the complexity of query evaluation in constraint database theory should be directed towards the identification of database and query classes that have a strongly improved complexity behavior. As a pattern for the development of such a theory, we may consider a new type of elimination algorithms which are based on the notion of system degree and use non-conventional data structures (see [2–4, 13, 15, 17, 20, 21, 24, 30, 34]).

This paper introduces a number of new concepts for constraint database theory that sometimes require certain notions from algebraic complexity theory, algebraic geometry and commutative algebra. These notions can be found in

standard textbooks, such as [10] (algebraic complexity theory), [1] (commutative algebra) and [35] (algebraic geometry). The reader only interested in database issues may read this paper while skipping these technical details (and in particular the proof sketch of Theorem 1 below).

The remainder of this paper is organized as follows. In Section 2, we discuss some properties of the classical data model for constraint databases and the impact of these properties on the complexity of query evaluation. In Section 3, we introduce a novel data model and the concept of sample point query. Then, we extend this concept to queries that return rationally parameterized algebraic families of polynomial functions as output. Furthermore, we introduce for this type of extended sample point queries the notion of a branching-parsimonious query evaluation algorithm. In Section 4, we argue that branching-parsimonious algorithms for extended sample point queries, associated to first-order formulas with a fixed number of quantifier alternations, require exponential sequential execution time. In Section 5, we discuss further directions of research in efficient query evaluation for constraint databases.

2 A Discussion of the Classical Data Model of Constraint Databases and Its Data Structures

In the current theory, a constraint database consists of a finite number of relations that are interpreted as geometric figures in some space $\mathbb{R}^n$. Quantifier-free first-order formulas over the reals are used to store these relations in the database [28,33]. The actual data structures needed to implement this data model[1] are only partially specified: the quantifier-free formulas are supposed to be given in disjunctive normal form and the polynomials appearing in them are supposedly given in dense or sparse representation. Geometric data in $\mathbb{R}^n$ are therefore described by expressions of the form

$$\bigvee_{i=1}^{d} \bigwedge_{j=1}^{c_i} p_{ij}(x_1, \ldots, x_n)\ \theta_{ij}\ 0,$$

where the $p_{ij}(x_1, \ldots, x_n)$ are polynomials with integer coefficients in the real variables $x_1, \ldots, x_n$ and where $\theta_{ij} \in \{\leq, \geq, <, >, =, \neq\}$. The sets that can be described in this way are known as *semi-algebraic sets* [9]. For example, the spatial figure consisting of the set of points on the northern hemisphere together with the points on the equator and the south pole of the unit sphere in the three-dimensional space $\mathbb{R}^3$ can be represented by the formula $(x_1^2 + x_2^2 + x_3^2 = 1 \wedge x_3 \geq 0) \vee (x_1 = 0 \wedge x_2 = 0 \wedge x_3 = -1)$.

The standard query language $\mathsf{FO}(+, \times, <, 0, 1, R_1, \ldots, R_s)$ for constraint databases is first-order logic over the reals augmented with relation predicates

[1] In the sequel, we use the terms *data model* and *data structure* in the way as above: the former refers to a conceptual notion whereas the latter refers to the structures that implement this notion.

$R_1, \ldots, R_s$ to address the relations that appear in the input database. When we want to emphasize that a relation predicate R has arity k, we write $R^{(k)}$. As an example of a first-order query, we take the $\mathsf{FO}(+, \times, <, 0, 1, R^{(3)})$-sentence $(\exists x_0)(\forall x_1)(\forall x_2)(\forall x_3)(R(x_1, x_2, x_3) \rightarrow x_1^2 + x_2^2 + x_3^2 < x_0^2)$ expresses that the three-dimensional spatial relation $R^{(3)}$ is bounded. Query evaluation based on quantifier elimination in elementary geometry guarantees that output relations defined by $\mathsf{FO}(+, \times, <, 0, 1, R_1, \ldots, R_s)$-queries can again be described within this data model.

2.1 Alternative Data Structures for Quantifier-Free Formulas

Let us now make a number of straightforward comments on the choice of this data model and data structure. It is clear that quantifier-free formulas give a effective way to check *membership* (e.g., to query outputs). In fact, the main aim of quantifier elimination in logic is to produce an effective method for checking membership. From the point of view of complexity of query evaluation, insisting on disjunctive normal form and dense or sparse representation of polynomials has a number of obvious disadvantages. For example, given a relation in disjunctive normal form, its complement may require a description that contains a number of disjuncts that grows exponentially in dimension of the ambient space (see [25] for upper and lower bounds on the number of disjuncts).

As a consequence of Bézout's theorem, the elimination of a block of n existential quantifiers in a formula containing polynomials of degree at most d, may produce output polynomials of degree d^n. For an illustration of this phenomenon, we refer to Example (2) in Section 2.2.

Consider now the family of queries, indexed by n $(n = 1, 2, \ldots)$, expressed by the formulas $\Phi_n(a_{11}, \ldots, a_{nn})$ given as

$$(\exists x_1) \cdots (\exists x_n)(R(a_{11}, \ldots, a_{nn}, x_1, \ldots, x_n) \wedge \bigvee_{i=1}^{n} x_i \neq 0)$$

in $\mathsf{FO}(+, \times, <, 0, 1, R^{(n^2+n)})$. When the query expressed by $\Phi_n(a_{11}, \ldots, a_{nn})$ is applied to the input database

$$A_n = \{(\alpha_{11}, \ldots, \alpha_{nn}, v_1, \ldots, v_n) \in \mathbb{R}^{n^2+n} \mid \bigwedge_{i=1}^{n} \sum_{j=1}^{n} \alpha_{ij} v_j = 0\},$$

we obtain, after plugging in the definition of A_n in the description of $\Phi_n(a_{11}, \ldots, a_{nn})$, a first-order sentence expressing that the determinant of the matrix $(\alpha_{ij})_{1 \leq i,j \leq n}$ is zero. This example shows that elimination polynomials may become dense even when their degrees are moderate. This makes the use of dense or sparse representation of polynomials is unsuitable for query evaluation.

The problem of the exploding representations leads to the idea of changing the data structure. This suggests the use of *arithmetic boolean circuits* for the representation of quantifier-free formulas. In order to illustrate this idea, let us

observe that the Gaussian elimination algorithm realizes an $O(n^3)$ size arithmetic boolean circuit (with divisions) which decides whether a given inhomogeneous linear $n \times n$ equation system has a solution.

2.2 The Influence of the Choice of Data Structure on Quantifier Elimination

We want to argue that complexity theory for geometric elimination requires *simultaneous* optimization of data structures *and* algorithms. To illustrate this point, let us consider the family of the first-order formulas

$$(\exists x_1)\cdots(\exists x_n)(x_1 = t+1 \wedge R(x_1, x_2) \wedge \cdots \wedge R(x_{n-1}, x_n) \wedge y = x_n^2),$$

indexed by n $(n = 1, 2, \ldots)$. When we apply these queries to the binary relation $A = \{(v_1, v_2) \in \mathbb{R}^2 \mid v_1^2 = v_2\}$, after plugging in the description of A in these formulas, we obtain the formulas $\Phi_n(t, y)$, given by

$$(\exists x_1)\cdots(\exists x_n)(x_1 = t+1 \wedge x_1^2 = x_2 \wedge \cdots \wedge x_{n-1}^2 = x_n \wedge y = x_n^2). \qquad (1)$$

The formula $\Phi_n(t, y)$ is logically equivalent to the following quantifier-free formula $\Psi_n(t, y)$:

$$y = \sum_{i=0}^{2^n} \binom{2^n}{i} t^i.$$

If we choose as data structure the dense or sparse representation of polynomials by their coefficients, then $\Phi_n(t, y)$ has length $O(n)$, whereas the length of $\Psi_n(t, y)$ is $O(2^n)$. When we use division-free arithmetic circuits (or straight-line programs) as data structure, then both $\Phi_n(t, y)$ and $\Psi_n(t, y)$ have length $O(n)$, since the polynomial $(t+1)^{2^n}$ can be evaluated using an iteration of n squarings (see [10] for a definition of the notions of arithmetic circuits and straight line program).

Unfortunately, one can show that general-purpose elimination algorithms, satisfying reasonable restrictions on the quality of the output representations, are not able to produce systematically polynomial size output descriptions. This complexity result holds for any continuous data structure, including the representation of polynomials by division-free arithmetic circuits (see [11]). With respect to upper complexity bounds for elimination algorithms, the state of the art is illustrated by the following simple knapsack-like example of an elimination problem. Consider the family of formulas

$$(\exists x_1)\cdots(\exists x_n)(R(x_1) \wedge \cdots \wedge R(x_n) \wedge y = u_1 x_1 + \cdots + u_n x_n),$$

indexed by n $(n = 1, 2, \ldots)$. When applying these queries to the unary constraint relation $A = \{(v) \in \mathbb{R} \mid v^2 - v = 0\}$, we obtain the first-order formulas $\Phi_n(y, u_1, \ldots, u_n)$:

$$(\exists x_1)\cdots(\exists x_n)(x_1^2 - x_1 = 0 \wedge \cdots \wedge x_n^2 - x_n = 0 \wedge y = u_1 x_1 + \cdots + u_n x_n). \quad (2)$$

This elimination problem has the following canonical quantifier-free output formula $\Psi_n(y, u_1, \ldots, u_n)$:

$$\prod_{(\varepsilon_1,\ldots,\varepsilon_n)\in\{0,1\}^n} (y - (\varepsilon_1 u_1 + \cdots + \varepsilon_n u_n)) = 0. \tag{3}$$

Using the dense or sparse representation of polynomials, the formula $\Psi_n(y, u_1, \ldots, u_n)$ takes $O(2^{n^2})$ space. However, checking membership by $\Psi_n(y, u_1, \ldots, u_n)$ requires only $O(2^n)$ arithmetical operations. The standard dense representation based general-purpose algorithms, cited in the constraint database literature [28], require at least $O(2^{n^2})$ sequential time to eliminate the n existential quantifiers in the input formula, whereas the use of the arithmetic boolean circuit representation leads to a sequential time complexity of $O(2^n)$ and therefore produces an exponential complexity gain (compare, e.g., [16, 22]). However, this example shows that in terms of the syntactical size of the input formula, the worst-case sequential time complexity of the elimination of a single existential quantifier block remains still exponential.

Let us make the following observations: specializing the free variables $u_1, \ldots, u_n$ and y of formula (2) into arbitrary non-negative integer values $\alpha_1, \ldots, \alpha_n$ and β, the resulting formula $\Phi_n(\beta, \alpha_1, \ldots, \alpha_n)$ becomes equivalent to the statement that there exists a set $I \subseteq \{1, \ldots, n\}$ with $\sum_{i\in I} \alpha_i = \beta$. Therefore, the closed formula $\Phi_n(\beta, \alpha_1, \ldots, \alpha_n)$ defines the knapsack problem given by the instance $\alpha_1, \ldots, \alpha_n$ and β.

On the other hand, specializing the free variables $u_1, \ldots, u_n$ into the integer values $2^0, \ldots, 2^{n-1}$ the elimination polynomial contained in the formula (3) becomes the well-known Pochhammer-Wilkinson polynomial

$$\prod_{0\le j<2^n} (y - j),$$

whose arithmetic circuit complexity is has been open since the times of Euler. In this context, let us mention that the polynomial $\prod_{0\le j<2^n}(y - \sqrt{j})$ requires $2^{\Omega(n)}$ arithmetic operations for its evaluation (see [22]). However, this polynomial has algebraic coefficients and therefore it cannot be produced by an elimination problem.

Altogether, the above examples reflect adequately the main complexity issues in elimination theory over the complex and real numbers both from the point of view of upper and lower sequential time complexity bounds (compare [11,14, 16]).

2.3 Where Does the Exponentiality of Elimination Algorithms Come from?

The subformula $x_1 = t + 1 \wedge x_1^2 = x_2 \wedge \cdots \wedge x_{n-1}^2 = x_n$ in (1) defines for each value of t just one point of the n-dimensional affine ambient space. On the other hand, the second system $x_1^2 - x_1 = 0 \wedge \cdots \wedge x_n^2 - x_n = 0$ in (2) has 2^n roots. This

observations may be paraphrased as follows: the "system degree" of formula (1) is one and that of formula (2) is 2^n.

In general, a given algebraic variety may be defined by different systems of polynomial equalities and inequalities. One may associate to each of these systems an invariant, called *system degree,* that is at least as large as the degree of the given variety and depends only on the polynomials and their order in the system but not on their representation (for formal definitions of different variants of the notion of system degree in the algebraic and the arithmetic setting see [3, 4, 13, 15, 20, 27]). The complexity issue of both of the above examples can be explained by a single general fact:

> *the sequential time complexity of elimination of an existential quantifier block is polynomial in the syntactic length of the input formula and its system degree.*

However, the second example shows that this fact does not prevent quantifier elimination to become exponential in the number of existentially quantified variables. This is due to the circumstance that the system degree may become exponential in the syntactical length of the input formula.

The system degree has shown to be a fruitful notion for the complexity analysis of quantifier elimination in elementary geometry. However, for query evaluation in constraint databases, the notion of system degree is still unsatisfactory since it is determined both by the query formula and the quantifier-free formulas describing the input database relations. It is a task for future constraint database research to develop a well-adapted complexity invariant in the spirit of the system degree in elimination theory.

On the other hand, the non-polynomial worst-case complexity character of the elimination of a single block of existential quantifiers can easily be deduced from the $\mathrm{P}_{\mathbb{R}} \neq \mathrm{NP}_{\mathbb{R}}$ conjecture in the Blum-Shub-Smale complexity model over the real numbers [8].

In Section 4, we show an exponential complexity lower bound, not depending on conjectures, in a slightly extended query model.

3 A Novel Data Model for Constraint Databases

We pass now to another shortcoming of the classical data model for constraint databases, i.e., of using quantifier-free formula in disjunctive normal form (encoded by whatever data structure): a quantifier-free formula in disjunctive normal form, describing, e.g., the output of a query, allows to answer the corresponding membership question, but it is does not allow an easy visualization of the output, i.e., this data model does not directly support data exploration and local visualization.

3.1 The Role of Sample Points in Constraint Databases

It is well known in elementary geometry that geometric figures can be given by equations and inequalities or in parametric form. In the first case, it is easy to

decide membership of a given point of the ambient space to the figure whereas it is difficult to produce even a single point belonging to the figure. When the figure is given in parametric form, the exact opposite is true.

A well-known algorithmic problem in elimination theory is the implicitation of a parametrically given geometric figure. The inverse problem is not always solvable in terms of polynomial or rational mappings. For example, the plane curve given by the equation $x^2 - y^3 = 0$ cannot be rationally parameterized. A reasonable half-way solution to overcome this problem is augmenting the implicit representation of a geometric figure with *sample points.*

The existing efficient elimination algorithms produce sample points as a natural byproduct (compare e.g. [3,4,6,18,23,32]). A famous example is CAD (see [12]), where sample points are used to determine the truth value of a given formula in a given cell. Therefore, when constraint database theory invokes one of the existing quantifier-elimination algorithms in query evaluation, it can incorporate this feature without loss of complexity.

A sample point of a geometric figure is not just any point belonging to that figure. In elimination theory, the notion of sample point has a specific meaning. Below, we give the definition of the notion of sample point in the context of semi-algebraic geometry (in the context of algebraic geometry over the complex numbers the corresponding definition is slightly different).

Definition 1. A *sample point* of a semi-algebraic set A in $\mathbb{R}^n$ is a quantifier-free first-order formula that defines (or encodes) exactly one point $(a_1, \ldots, a_n)$ of A. We require that this formula allows us to determine, for any polynomial $p \in \mathbb{Z}[x_1, \ldots, x_n]$, the sign of $p(a_1, \ldots, a_n)$ in a finite number of steps using only arithmetic operations and comparations in $\mathbb{Q}$. □

We will also use the term sample point to refer to the geometric point defined by this quantifier-free formula. A *sample point query* is a computable function that returns on input a semi-algebraic set, represented by a formula, at least one sample point of this set (e.g., one for each of its connected components).

Sample points are computable from a given first-order description of the set A. As the following proposition shows, sample points are also first-order definable in the sense that there is a expression that depends on the dimension of the ambient space and that returns as output a point that allows a description in the sense of Definition 1.

Proposition 1. *For any integer $n \geq 1$, there exists a formula $\sigma_n(x_1, \ldots, x_n)$ in $\mathsf{FO}(+, \times, <, 0, 1, R^{(n)})$, such for any non-empty semi-algebraic subset A of $\mathbb{R}^n$, the set $\{(a_1, \ldots, a_n) \in \mathbb{R}^n \mid (A, a_1, \ldots, a_n) \models \sigma_n(x_1, \ldots, x_n)\}$ consists of exactly one point that belongs to A and that satisfies the requirements of a sample point as stated in Definition 1.*

Proof. We prove this property by induction on n. For $n = 1$, the following procedure, returning a number x_1, can be expressed by a formula $\sigma_1(x_1)$ in $\mathsf{FO}(+, \times, <, 0, 1, R^{(1)})$: if ∂R is empty[2] (in particular if R equals $\mathbb{R}$), then return

[2] We denote the topological border of a set R by ∂R.

the origin 0; else, if ∂R contains exactly one element p, then return the minimum of the set $\{p-1, p, p+1\} \cap R$; else, if ∂R contains at least two elements, let p be the smallest and q be the one but smallest element of ∂R and return the minimum of the set $\{p-1, p, \frac{p+q}{2}\} \cap R$.

Clearly, for any non-empty semi-algebraic subset A of $\mathbb{R}$, the set $\{a \in \mathbb{R} \mid (A, a) \models \sigma_1(x_1)\}$ contains a single point that belongs to A. Since the above procedure is $\mathsf{FO}(+, \times, <, 0, 1, R^{(1)})$-expressible, the defined point is algebraic and algebraic points satisfy the requirements of a sample point as stated in Definition 1.

Let $n > 1$ and assume the property holds for all m, $1 \leq m < n$. Consider the following procedure that, on input a semi-algebraic subset R of $\mathbb{R}^n$, returns an n-dimensional real vector: let $\pi_1(R)$ be the projection of the subset R of $\mathbb{R}^n$ onto the first coordinate x_1 and let α_1 be the only element of $\mathbb{R}$ satisfying $\sigma_1(x_1)$ when applied to $\pi_1(R)$; let $(\alpha_2, \ldots, \alpha_n)$ be the unique vector defined by $\sigma_{n-1}(x_2, \ldots, x_n)$, when applied to the projection on the coordinates $(x_2, \ldots, x_n)$ of the intersection of R and the hyperplane $x_1 = \alpha_1$; return $(\alpha_1, \alpha_2, \ldots, \alpha_n)$.

It should be clear that this procedure can be expressed by a formula $\sigma_n(x_1, \ldots, x_n)$ in $\mathsf{FO}(+, \times, <, 0, 1, R^{(n)})$. Clearly, this procedure, when applied to a non-empty semi-algebraic subset A of $\mathbb{R}^n$, defines a single point of A. The same argument as above shows that this point satisfies the requirements of a sample point as stated in Definition 1. □

A well-known encoding of sample points is based on Thom's lemma, saying that any real algebraic number may be encoded by listing sign conditions on a suitable polynomial $p \in \mathbb{Z}[x]$ and all its derivatives.

Optimization problems. An illustration of the relevance of sample points can be found in optimization theory, where one typically asks the following three questions:

1. Given a system of linear inequalities $\sum_{j=1}^{n} a_{ij} x_j \geq b_i$ $(1 \leq i \leq n)$, determine whether it has a solution, i.e., decide whether the formula
$$(\exists x_1) \cdots (\exists x_n) \bigwedge_{i=1}^{m} \sum_{j=1}^{n} a_{ij} x_j \geq b_i$$
is true.
2. If a system of linear inequalities $\sum_{j=1}^{n} a_{ij} x_j \geq b_i$ $(1 \leq i \leq n)$ has a non-empty solution set V, decide whether a given affine target function f defined by $(x_1, \ldots, x_n) \mapsto \sum_{i=1}^{n} c_i x_i + d$ reaches a finite maximum on V.
3. If f reaches such a maximum on V, give an example of a point in V that realizes this maximum.

Clearly, given the linear equality system and the affine function in a suitable form as a relation (we refer to the encoding of a system of equations in Section 2.1 as an example of such a representation), Problems 1 and 2 can readily be seen as instances of constraint database queries. Furthermore, these queries

are first-order expressible. By Proposition 1, also Problem 3 can be expressed in first-order logic. Problem 3 is an example of a sample point query. These problems show that optimization problems naturally belong to constraint database theory. They also show that enriching implicit descriptions of constraint data with sample points, facilitates the answering of optimization queries.

3.2 A Novel Data Model for Constraint Databases with Cell Decompositions and Sample Points

In order to make constraint relations more explicit, we have argued that adding sample points to first-order formulas is important. Furthermore, it can be desirable that the data model supports an exploration of the neighborhood of such a sample point. This can be achieved by representing constraint relations by cell decompositions consisting of cells which are open subsets of smooth real algebraic varieties together with suitable sample points. In this sense, we are going to propose a novel data model for constraint databases. On the other hand, existing efficient elimination procedures produce in a natural way output representations which fit perfectly well in this data model.

In this manner, starting from any sample point, its neighborhood within its cell can be explored.

Let us now make our ideas more precise.

Definition 2. Let A be a semi-algebraic set contained in the n-dimensional affine space $\mathbb{R}^n$. A *cell decomposition of* A is a finite family $\mathcal{F}_1, \ldots, \mathcal{F}_m$ of data of the following kind. For each k, $1 \leq k \leq m$, $\mathcal{F}_k$ is a list consisting of non-zero polynomials of $\mathbb{Q}[x_1, \ldots, x_n]$, namely $f_1^{(k)}, \ldots, f_{s_k}^{(k)}$, with $0 \leq s_k \leq n$, $g_1^{(k)}, \ldots, g_{t_k}^{(k)}$, and ρ_k, for $r_k := n - s_k$ a rational $r_k \times n$ matrix M_k of maximal rank and a finite family of sample points $x_1^{(k)}, \ldots, x_{q_k}^{(k)}$ of A such that the following conditions are satisfied:

(i) the equations $f_1^{(k)} = 0, \ldots, f_{s_k}^{(k)} = 0$, intersect transversally in all its common real zeroes where the polynomial ρ_k does not vanish (and such real zeroes exist);

(ii) for $(y_1, \ldots, y_{r_k})^\tau = M_k(x_1, \ldots, x_n)^\tau$ the polynomial ρ_k belongs to $\mathbb{Q}[y_1, \ldots, y_{r_k}]$ and the linear forms $y_1, \ldots, y_{r_k}$ induce a finite unramified morphism from the complex variety defined by the equations $f_1^{(k)} = 0, \ldots, f_{s_k}^{(k)} = 0$ in $\mathbb{C}^n \setminus \{\rho_k \neq 0\}$ onto $\mathbb{C}^{r_k} \setminus \{\rho_k \neq 0\}$;

(iii) the semi-algebraic set (cell) A_k, defined by the conjunction $f_1^{(k)} = 0 \wedge \cdots \wedge f_{s_k}^{(k)} = 0 \wedge g_1^{(k)} > 0 \wedge \ldots \wedge g_{t_k}^{(k)} > 0 \wedge \rho_k \neq 0$, is non-empty and contained in A and $\dim_x A_k = r_k$ holds for any point x of A_k;

(iv) the sample points $x_1^{(k)}, \ldots, x_{q_k}^{(k)}$ belong to A_k and each connected component of A_k contains at least one of these sample points;

(v) the semi-algebraic set A is the union of all cells A_k $(1 \leq k \leq m)$. □

Remark that the well-known CAD algorithm produces a cell decomposition in the sense of the above definition [12]. However, it usually produces adjacent cells that can be merged still resulting in a cell decomposition.

In order to illustrate the concept of cell decomposition, let us consider the closed unit disk A of the real plane $\mathbb{R}^2$, defined by the inequality

$$x_1^2 + x_2^2 \leq 1.$$

The semi-algebraic set A may be decomposed into three cells, namely the topological interior A_1 of A and the curves $A_2 := \partial A \setminus \{(1,0),(-1,0)\}$ and $A_3 := \partial A \setminus \{(0,1),(0,-1)\}$ (here ∂A denotes the topological border of A).

The cells A_1, A_2, A_3 may be defined by the following data sets $\mathcal{F}_1, \mathcal{F}_2, \mathcal{F}_3$. We first introduce $\mathcal{F}_1$. Let $s_1 := 0$, $t_1 := 0$, $q_1 := 1$, $g_1^{(1)} := 1 - x_1^2 - x_2^2$ and $x_1^{(1)} := (0,0)$. One verifies immediately that $\mathcal{F}_1$ defines the interior A_1 of A and that A_1 is connected and is containing the origin $x_1^{(1)}$ of $\mathbb{R}^2$ as sample point.

Next, we introduce the data set $\mathcal{F}_2$. Let $s_2 := 1$, $t_2 := 1$, $q_2 := 2$, $f_1^{(2)} := x_1^2 + x_2^2 - 1$, $M_2 := (1,0)$, $\rho_2 := 1 - x_1^2$, $g_1^{(2)} := 1 - x_1^2$ and $x_1^{(2)} := (0,1)$ and $x_2^{(2)} := (0,-1)$. Obviously, $\mathcal{F}_2$ defines the cell $A_2 = \partial A \setminus \{(1,0),(-1,0)\}$ which contains two connected components with the sample points $x_1^{(2)}$ and $x_2^{(2)}$.

Finally, we introduce the data set $\mathcal{F}_3$. Let $s_3 := 1$, $t_3 := 1$, $q_3 := 2$, $f_1^{(3)} := x_1^2 + x_2^2 - 1$, $M_3 := (0,1)$, $\rho_3 := 1 - x_2^2$, $g_1^{(3)} := 1 - x_2^2$ and $x_1^{(3)} := (1,0)$ and $x_2^{(3)} := (-1,0)$. Similarly as before one verifies that $\mathcal{F}_3$ defines the cell $A_3 = \partial A \setminus \{(0,1),(0,-1)\}$ which contains two connected components with the sample points $x_1^{(3)}$ and $x_2^{(3)}$.

In view of Subsection 2.1, we think cell decompositions suitably encoded by arithmetic boolean circuits. In particular, we think the polynomials $f_1^{(k)}, \ldots, f_{s_k}^{(k)}$, $g_1^{(k)}, \ldots, g_{t_k}^{(k)}$, and ρ_k represented by division-free arithmetic circuits. Using numerical procedures, it is now clear that cell decompositions allow local visualization of the semi-algebraic sets they represent. In the following, we discuss our new data model rather informally.

The cell decompositions we are proposing as data model for constraint databases should come as close as possible to stratifications. Geometrically speaking, an arbitrary cell decomposition of a semi-algebraic set does not capture adequately the phenomenon of cell degeneration, and, algorithmically speaking, an *arbitrary* arithmetic boolean circuit representation of this cell decomposition may introduce spurious branchings, mainly by the use of membership tests based on algorithms which require divisions. This leads us to the requirement that the arithmetic boolean circuit representations of cell decompositions should be *branching-parsimonious* (see [11,14]). In particular, we require that our arithmetic boolean circuits do not contain branchings where traditional or alternative but still efficient membership tests do not introduce them. For instance, testing membership to a determinantal variety may be performed using an arithmetic boolean circuit with branchings implementing the traditional Gaussian elimination method. On the other hand, Berkowitz' polynomial time algorithm for the computation of the determinant allows an efficient alternative and division-free representation of the same variety [7]. In this sense, Berkowitz'algorithm is a branching-parsimonious (in fact, branching-free) substitute for Gaussian elimination.

We call an *algorithm* for query evaluation *branching-parsimonious* if on any input a branching-parsimonious representation of its output is returned.

At the present we have no general overview of all geometric situations where branchings can be avoided and therefore we do not dispose of a rigorous mathematical definition of the intuitive concept of a branching-parsimonious algorithm. Nevertheless, in this paper we shall make only use of this concept in the very narrow situation of branching-free output representations of generalized sample point queries, which will be introduced and (rigorously) defined at the end of this section.

If a database is given in traditional form (by a user), namely by polynomial constraints (e.g., encoded by arithmetic boolean circuits), one may ask for more suitable alternative representations in order to improve the complexity of query evaluation. Such an improved representation may be a branching-parsimonious cell decomposition of the constraint relations, as described before. The generation of such a cell decomposition may be considered as a pre-processing of the data before actually storing it in the database.

The example of finite databases. Let us explain the idea behind this observation in the case of a finite database containing just one relation D. For this purpose we shall make free use of the main complexity results of [13,15,17,21]. Let us suppose that the domain of D is described by a conjunction of the form

$$f_1 = 0 \wedge \cdots \wedge f_s = 0 \wedge g \neq 0,$$

with $f_1, \ldots, f_s, g \in \mathbb{Q}[x_1, \ldots, x_n]$, and that this conjunction defines a finite set V of points of $\mathbb{C}^n$ having a non-empty intersection with $\mathbb{R}^n$. Suppose furthermore that the conjunction $f_1 = 0 \wedge \cdots \wedge f_s = 0 \wedge g \neq 0$ is encoded by a division-free arithmetic boolean circuit of size at most L that contains only decision gates for equality. Let d be an upper bound for the degrees of the polynomials $f_1, \ldots, f_s$ and δ be the system degree associated to the conjunction $f_1 = 0 \wedge \cdots \wedge f_s = 0 \wedge g \neq 0$ (thus we have $\#V \leq \delta \leq d^n$). Under some very weak conditions on the data $f_1, \ldots, f_s$ and g (e.g., we may require $s := n$ and that $f_1, \ldots, f_n$ form a reduced regular sequence outside the locus $\{g = 0\}$) one can find in sequential time $L(nd\delta)^{O(1)}$ a rational linear form $u = \lambda_1 x_1 + \cdots \lambda_n x_n$ with $\lambda_1 \neq 0$ and univariate polynomials $q, v_2, \ldots, v_n \in \mathbb{Q}[t]$ (where t is a new variable) satisfying the following conditions:

(i) $\deg q \leq \#V$, $\deg v_2 \leq \#V, \ldots, \deg v_n \leq \#V$;
(ii) q has no multiple zeroes;
(iii) V is definable by the conjunction $q(u) = 0 \wedge x_2 = v_2(u) \wedge \cdots \wedge x_n = v_n(u)$.

One sees immediately that the polynomials $q(u), x_2 - v_2(u), \ldots, x_n - v_n(u)$ yield a description of $D = \mathbb{R}^n \cap V$ by a single cell in the sense of Definition 2. Moreover, for $v_1 := \frac{1}{\lambda_1}(t - \lambda_2 v_2 - \cdots - \lambda_n v_n)$ we obtain the following *parametric* description of the complex and real varieties V and D respectively. Namely, $V := \{(v_1(\tau), \ldots, v_n(\tau)) \mid q(\tau) = 0, \tau \in \mathbb{C}\}$ and $D := \{(v_1(\tau), \ldots, v_n(\tau)) \mid q(\tau) = 0, \tau \in \mathbb{R}\}$.

Evaluation of purely existential active domain FO(+, ×, <, 0, 1)-queries over the database D may now be reduced to greatest common divisor computations between univariate polynomials over $\mathbb{Q}$ (here we suppose that all polynomials occurring in the query are given by division-free arithmetic boolean circuits). It is possible to associate to each such query a degree, depending only on the occurrences of relation predicate addressing D in the query.

Evaluation of such a query becomes now linear in the syntactic query size and polynomial in its degree.

The example of rationally parameterized families of polynomial functions. A particular instance of interest is the case that the semi-algebraic set A is contained in $\mathbb{R}^{m+n+1}$ and represents a rational family of polynomial functions from $\mathbb{R}^n$ to $\mathbb{R}$. To be more precise, let $\pi : \mathbb{R}^{m+n+1} \to \mathbb{R}^m$ be the canonical projection of any point of $\mathbb{R}^{m+n+1}$ on its first m coordinates. Suppose that A is non-empty and that for any $u = (u_1, \ldots, u_m) \in \pi(A)$ the semi-algebraic set $(\{u\} \times \mathbb{R}^{n+1}) \cap A$ is the graph of an n-variate polynomial $f_u \in \mathbb{R}[x_1, \ldots, x_n]$. It is a natural extension of our previously introduced sample-point query to ask for a procedure which enables us for each $u \in \pi(A)$ and each $x \in \mathbb{R}^n$ to compute the value of $f_u(x)$. The output of such a procedure may be a purely existential prenex first-order formula in the free variables $u_1, \ldots, u_m$ and $x_1, \ldots, x_n$ which represents for each $u \in \pi(A)$ a division-free arithmetic circuit which evaluates the polynomial f_u (observe that there exists a uniform degree bound for all these polynomials). One easily verifies that all our requirements on the semi-algebraic set A, except that of the polynomial character of the function represented by the graph $(\{u\} \times \mathbb{R}^{n+1}) \cap A$, are first-order definable over the reals. Nevertheless, over the complex numbers, when A is a constructible (i.e., a first-order definable subset of $\mathbb{C}^{m+n+1}$), all these requirements are first-order expressible. This leads us to a new type of computable queries which return on input a semi-algebraic or constructible set A as above and an element $u \in \pi(A)$, a first-order formula which represents a division-free arithmetic circuit evaluating the polynomial f_u. Uniformity of query evaluation with respect to u is expressed by the requirement that the terms contained in this formula have to depend *rationally* on u.

In the following, we shall refer to this type of queries as *extended sample point queries.* We shall refer to $u_1, \ldots, u_m$ as the *parameters* and to $x_1, \ldots, x_n$ as the *variables* of the query.

Suppose now that A is a constructible subset of $\mathbb{C}^{m+n+1}$ with irreducible Zariski-closure. Let V be the Zariski-closure of $\pi(A)$. Then V is an irreducible affine subvariety of $\mathbb{C}^m$. We denote the function field of V by K. It is not difficult to see that for generically chosen parameter points $u \in \pi(A)$, the extended sample point query associated to A can be realized by a greatest common divisor computation in the polynomial ring $K[x_1, \ldots, x_n]$.

Variables versus parameters. The previous example is motivated by spatial data that come from physical observation and are only known with uncertainty. Another motivation comes from parametric optimization. The optimization problems described in Section 3.1 can also be studied in parametric form, i.e., in the

case where the linear inequalities and the target function contain coefficients that depend on a time parameter [5], In this case, an optimum is not an arbitrary set of sample points but an analytic (or at least continuous) function which depends on a time parameter.

We are now going to explain why we distinguished between the parameters $u_1, \ldots, u_m$ and the variables $x_1, \ldots, x_n$ in our discussion of rationally parameterized families of polynomial functions. In the example above, let $\Phi(u_1, \ldots, u_m, x_1, \ldots, x_m, y)$ be a quantifier-free formula which defines the semi-algebraic or constructible set A. Let us suppose that Φ contains a subformula $\Psi(u_1, \ldots, u_m)$ which expresses an internal algebraic dependency between the parameters $u_1, \ldots, u_m$. With respect to the variables $x_1, \ldots, x_n$ there is no such subformula contained in Φ. For the sake of simplicity, we shall suppose that Ψ defines the set $\pi(A)$ and that there exists a formula $\Omega(u_1, \ldots, u_m, x_1, \ldots, x_n, y)$ such that $\Phi(u_1, \ldots, u_m, x_1, \ldots, x_m, y)$ can be written as

$$\Psi(u_1, \ldots, u_m) \wedge \Omega(u_1, \ldots, u_m, x_1, \ldots, x_m, y).$$

In Section 4, we shall meet natural examples of parameterized algebraic families of polynomial functions where the formula Ψ becomes of uncontrolled size and is of few interest, whereas the formula Ω becomes the relevant part of the output information of a suitable elimination algorithm. This situation occurs for instance when the points u of $\mathbb{R}^m$ satisfying the formula Ψ are given in parametric form (i.e., when they are image points of some polynomial or rational map coming from some affine source space). In this case, we are only interested in the subformula Ω, since points $u \in \mathbb{R}^m$ satisfying Ψ can easily be produced in sufficient quantity. In subsequent queries, $x_1, \ldots, x_n$ may appear as bounded variables, whereas the parameters $u_1, \ldots, u_m$ are not supposed to be subject to quantification. The example of the $u_1, \ldots, u_m$ expressing uncertainty in physical spatial data illustrates this. These different rôles motivate us to distinguish between $u_1, \ldots, u_m$ and $x_1, \ldots, x_n$ and to call them parameters and variables, respectively.

The branching-parsimonious algorithmic model. In the model, that we are going to use in the sequel, parameters and variables receive a different treatment. (Free) variables may be specialized into arbitrary real (or complex) values, whereas the specialization of parameters may be subject to certain restrictions. In the above example of a rationally parameterized family of polynomial functions, the n-tuple of variables $(x_1, \ldots, x_n)$ may be specialized into any point of the affine space $\mathbb{R}^n$ (or $\mathbb{C}^n$), whereas the m-tuple of parameters $(u_1, \ldots, u_m)$ may only be specialized into points satisfying $\Psi(u_1, \ldots, u_m)$, i.e., into points belonging to $\pi(A)$. Once the n-tuple of variables $(x_1, \ldots, x_n)$ is specialized into a point of the corresponding affine space, this point cannot be modified anymore. However, we allow infinitesimal modifications of a given specialization of the m-tuple of parameters $(u_1, \ldots, u_m)$ within the domain of definition determined by the formula Ψ. In the branching-parsimonious model, we require that an arithmetic boolean circuit which represents the semi-algebraic or constructible set A does not contain divisions which involve the variables $x_1, \ldots, x_n$. Similarly, for a given point

$u \in \pi(A)$, we require that the arithmetic circuit representing the polynomial f_u is division-free. However, divisions by algebraic expressions in the parameters $u_1, \ldots, u_m$ are sometimes unavoidable (e.g., in the case of parametric greatest common divisor computations; see [11,14]). Therefore, we allow certain limited divisions by algebraic expressions which depend only on the parameters $u_1, \ldots, u_m$. More precisely, we allow that the arithmetic boolean circuits representing the set A or the output of the corresponding extended sample point query computes certain, but not arbitrary, rational functions in the parameters $u_1, \ldots, u_m$, called *scalars* of the circuit. However, we do not allow the division of a positive-degree polynomial in the variables $x_1, \ldots, x_n$ by a non-constant scalar. In the above sense, we require for our branching-parsimonious algorithmic model that arithmetic boolean circuits are *essentially division-free with respect to variables* (see [11,14] for a precise definition).

Branching-free output representations of extended sample point queries. Since we allow certain infinitesimal modifications of the parameters $u_1, \ldots, u_m$ within their domain of definition, we sometimes may replace divisions (and corresponding branchings) by limit processes in the spirit of L'Hôpital's rule. It is possible to mimic algebraically this kind of limit process by places (see [29] for the notion of place and [11,14] for motivations of this idea).

Branchings corresponding to divisions can trivially be avoided by restricting input data. Therefore a meaningful notion of branching-parsimonious (or branching-free) algorithm requires the consideration of Zariski-closures of input data sets. This may partially explain the rather technical assumptions and tools in the following ad hoc definition of a branching-free representation of the output of an extended sample point query.

Suppose now that in the example above A is a constructible subset of $\mathbb{C}^{m+n+1}$ with irreducible Zariski-closure B. Let V be the Zariski-closure of $\pi(A)$ in $\mathbb{C}^m$. Then V is an irreducible affine variety whose function field we denote by K. Moreover, the irreducible affine variety B is birationally equivalent to $V \times \mathbb{C}^n$. Suppose furthermore that $\pi(B) = V$ holds and that B represents a rationally parameterized family of polynomial functions which extends the family represented by A. Then we say that the extended sample point query associated with A admits a *branching-free output representation* if there exists an essentially division-free, single-output arithmetic circuit β with inputs $x_1, \ldots, x_n$ and scalars $\theta_1, \ldots, \theta_s \in K$ satisfying the following conditions:

(i) for any point $u \in \pi(A)$ where the rational functions $\theta_1, \ldots, \theta_s$ are defined, the division-free arithmetic circuit, obtained from β by specializing the scalars $\theta_1, \ldots, \theta_s$ into the complex values $\theta_1(u), \ldots, \theta_s(u)$, evaluates the polynomial f_u;

(ii) for any point $u \in V$ and any place $\varphi : K \to \mathbb{C} \cup \{\infty\}$ whose valuation ring extends the local ring of the affine variety V at the point u, the values $\varphi(\theta_1), \ldots, \varphi(\theta_s)$ are finite and uniquely determined by u (therefore we shall write $\theta_1(u) := \varphi(\theta_1), \ldots, \theta_s(u) := \varphi(\theta_s)$).

Let an arithmetic circuit β be given as above. Then we call β a branching-free representation of the output of the extended sample point query associated to A.

Observe that the output of the circuit β represents a polynomial belonging to $K[x_1, \ldots, x_n]$ whose coefficients satisfy condition (ii). Moreover, the arithmetic circuit β constitutes a division-free representation of the extended sample point query associated to the Zariski-closure B of A. Finally, let us remark that for any $u \in V$, $x \in \mathbb{C}^n$ and $y \in \mathbb{C}$, the point (u, x, y) belongs to B if and only if the circuit β_u, obtained from β by replacing the scalars $\theta_1, \ldots, \theta_s$ by the complex numbers $\theta_1(u), \ldots, \theta_s(u)$, computes on input x the output y.

We require that a *branching-parsimonious query evaluation algorithm* produces a branching-free output representation of the given extended sample point query if the query admits such a representation.

Let us also observe that extended sample point queries appear in a natural way if we apply the constraint database concept to data processing in the context of approximation theory and functional analysis.

4 A Lower Complexity Bound for Extended Sample Point Queries

In this section, we restrict our attention to constraint databases defined in the language FO(+, ×, 0, 1, =) over the complex numbers. We shall consider two ternary relational predicates, namely $S(v_1, v_2, w)$ and $P(v_1, v_2, w)$. Our query language will therefore be FO(+, ×, 0, 1, =, *S*, *P*). Let L, n be given natural numbers and let $r := (L + n + 1)^2$. For any polynomial $f \in \mathbb{C}[x_1, \ldots, x_n]$, we denote by $L(f)$ the minimal non-scalar size of all division-free arithmetic circuits with inputs $x_1, \ldots, x_n$ and scalars from $\mathbb{C}$ which evaluate the polynomial f. Let

$$W_{L,n} := \{f \in \mathbb{C}[x_1, \ldots, x_n] \mid L(f) \leq L\}.$$

One sees easily that all polynomials contained in $W_{L,n}$ have degree at most 2^L and that $W_{L,n}$ forms a $\mathbb{Q}$-definable object class which has a $\mathbb{Q}$-definable holomorphic encoding by the continuous data structure $\mathbb{C}^r$. Observe that Zariski-closure $\overline{W_{L,n}}$ of $W_{L,n}$ is a $\mathbb{Q}$-definable, absolutely irreducible algebraic variety consisting of the polynomials of $\mathbb{C}[x_1, \ldots, x_n]$ which have approximate complexity at most L. Moreover, the affine variety $\overline{W_{L,n}}$ forms a cone in its ambient space (i.e., for any $\lambda \in \mathbb{C}$ we have $\lambda \overline{W_{L,n}} \subseteq \overline{W_{L,n}}$). For details on complexity and data structure models we refer to [10, 11].

Let $z_1, \ldots, z_r$ and y be new variables. Choose now a directed acyclic graph $\mathcal{D}_{L,n}$ representing a generic, division-free arithmetic circuit with input nodes $x_1, \ldots, x_n$, output node y and scalar nodes $z_1, \ldots, z_r$ such that any polynomial of $W_{L,n}$ may be evaluated by the division-free arithmetic circuit obtained from $\mathcal{D}_{L,n}$ by a suitable specialization of the parameters $z_1, \ldots, z_r$ into complex values. Without loss of generality, we may assume that the number of internal nodes of $\mathcal{D}_{L,n}$ is of order $O((L + n)^2)$. Translating the structure of the directed acyclic graph $\mathcal{D}_{L,n}$ into first-order logic one infers easily a formula

$\Psi_{L,n}(S, P, z_1, \ldots, z_r, x_1, \ldots, x_n, y)$ in the free variables $z_1, \ldots, z_r, x_1, \ldots, x_n, y$ of the query language FO(+, ×, 0, 1, =, *S*, *P*) such that $\Psi_{L,n}(S, P)$ satisfies the following conditions.

(i) $\Psi_{L,n}(S, P)$ is prenex, purely existential and of length $O((L+n)^2)$;
(ii) interpreting the predicates S and P in $\Psi_{L,n}$ as the graphs of the addition and the multiplication of complex numbers, and specializing the variables $z_1, \ldots, z_r$ into the complex numbers $\zeta_1, \ldots, \zeta_r$, the formula $\Psi_{L,n}(S, P, \zeta_1, \ldots, \zeta_r, x_1, \ldots, x_n, y)$ describes the graph of the polynomial of $\mathbb{C}[x_1, \ldots, x_n]$ computed by the arithmetic circuit, obtained from $\mathcal{D}_{L,n}$ by specializing the scalars $z_1, \ldots, z_r$ into $\zeta_1, \ldots, \zeta_r$.

Let $m := 4(L+n)^2 + 2$. From [11], Corollary 3 (see also [14, Lemma 4]) we deduce that there exists an identification sequence $\gamma_1, \ldots, \gamma_m \in \mathbb{Q}^n$ for the object class $\overline{W_{L,n}}$. Let $\Delta_{L,n}(S, P)$ be a closed FO(+, ×, 0, 1, =, *S*, *P*)-formula saying that S and P are the graphs of two binary operations which map $\mathbb{C}^2$ into $\mathbb{C}$, that $\gamma_1, \ldots, \gamma_n$ is an identification sequence for the object class of applications from $\mathbb{C}^n$ to $\mathbb{C}$, defined by the FO(+, ×, 0,1, =, *S*, *P*)-formula $\Psi_{L,n}(S, P)$ and that this object class is not empty. Without loss of generality, we may assume that $\Delta_{L,n}(S, P)$ has length $O((L+n)^2)$ and is prenex with a fixed number of quantifier alternations (which is independent of L and n).

We consider now the FO(+, ×, 0, 1, =, *S*, *P*)-formulas $\Phi_{L,n}(S, P, u_1, \ldots, u_r, x_1, \ldots, x_n, y)$ defined by

$$(\exists z_1) \cdots (\exists z_r)(\Psi_{L,n}(S, P, z_1, \ldots, z_r, x_1, \ldots, x_n, y) \wedge \bigwedge_{1 \le k \le m} \Psi_{L,n}(S, P, z_1, \ldots, z_r, \gamma_k, u_k)) \wedge \Delta_{L,n}(S, P)$$

and $\Omega_{L,n}(S, P, u_1, \ldots, u_r)$ defined by

$$(\exists z_1) \cdots (\exists z_r)(\bigwedge_{1 \le k \le m} \Psi_{L,n}(S, P, z_1, \ldots, z_r, \gamma_k, u_k)) \wedge \Delta_{L,n}(S, P).$$

Without loss of generality, we may assume that $\Phi_{L,n}(S, P)$ and $\Omega_{L,n}(S, P)$ are prenex formulas of length $O((L+n)^2)$ having a *fixed* number of quantifier alternations and containing the free variables $u_1, \ldots, u_m, x_1, \ldots, x_n, y$ and $u_1, \ldots, u_m$, respectively. Let $\pi : \mathbb{C}^{m+n+1} \to \mathbb{C}^m$ be the canonical projection which maps each point of $\mathbb{C}^{m+n+1}$ on its first m coordinates and let D be a constraint database over the schema *(S, P)* over the complex numbers. Suppose that D satisfies the formula $\Delta_{L,n}(S, P)$. With respect to the database D, the formula $\Phi_{L,n}(S, P, u_1, \ldots, u_r, x_1, \ldots, x_n, y)$ defines a non-empty constructible subset $A_{L,n}(D)$ of $\mathbb{C}^{m+n+1}$ and the formula $\Omega_{L,n}(S, P, u_1, \ldots, u_r)$ defines the set $\pi(A_{L,n}(D))$. Moreover, for any $u \in \pi(A_{L,n}(D))$, the formula $\Phi_{L,n}(S, P, u, x_1, \ldots, x_n, y)$ describes the graph of a n-variate polynomial map. Therefore, it makes sense to consider, for any natural numbers n and L, the generalized sample point query associated to the formula $\Phi_{L,n}(S, P, u_1, \ldots, u_n, x_1, \ldots, x_n, y)$. Suppose now that there is given a *branching-parsimonious* procedure $\mathcal{P}$ which evaluates this family of extended sample point queries. In the sequel we are going

to analyze the complexity behaviour of $\mathcal{P}$ for this query on the particular input database D, where S and P are interpreted as the graphs of the sum and the product of complex numbers.

We are now able to state the main complexity result of this paper.

Theorem 1. *Let notations and assumptions be as before. Then the branching-parsimonious procedure $\mathcal{P}$ requires sequential time $2^{\Omega(n)}$ in order to evaluate on input the database D the extended sample point query associated to the size $O(n^2)$ first-order formula $\Phi_{n,n}(S,P)$. In particular, extended sample point queries associated to first-order formulas with a fixed number of quantifier alternations cannot be evaluated by branching-parsimonious procedures in polynomial time.*

Proof. (Sketch). The arguments we are now going to use follow the general lines of the proofs of [14, Theorem 5] and [11, Theorem 4]. For a complete proof of Theorem 1 we refer to the full version of this paper.

For the moment let us fix the integer parameters L and n.

Observe that the closed formula $\Delta_{L,n}(S,P)$ is valid on the database D. Therefore the constructible set $A_{L,n} := A_{L,n}(D)$ is nonempty.

Let $B_{L,n}$ and $V_{L,n}$ be the Zariski-closures of $A_{L,n}$ and $\pi(A_{L,n})$ in $\mathbb{C}^{m+n+1}$ and $\mathbb{C}^m$, respectively, and let $\mu_{L,n} := \overline{W}_{L,n} \to \mathbb{C}^m$ be the morphism of $\mathbb{Q}$-definable affine varieties defined for $f \in \overline{W}_{L,n}$ by

$$\mu_{L,n}(f) := (f(\gamma_1), \ldots, f(\gamma_m)).$$

Observe that $B_{L,n}$ and $V_{L,n}$ are $\mathbb{Q}$-definable absolutely irreducible affine varieties and that the image of $\mu_{L,n}$ is $V_{L,n}$. It turns out that $\mu_{L,n} : \overline{W}_{L,n} \to V_{L,n}$ is a finite, bijective and birational morphism of affine varieties. This implies $\pi(B_{L,n}) = V_{L,n}$ and that $B_{L,n}$ represents a rationally parameterized family of polynomial functions which extends the family represented by $A_{L,n}$.

Let $K_{L,n}$ be the function field over $\mathbb{C}$ of the absolutely irreducible variety $V_{L,n}$ and let $R_{L,n}$ be the $\mathbb{C}$-algebra of all rational functions θ of $K_{L,n}$ such that for any point $u \in V_{L,n}$ and any place $\varphi : K_{L,n} \to \mathbb{C} \cup \{\infty\}$ whose valuation ring extends the local ring of the affine variety $V_{L,n}$ at the point u, the value $\varphi(\theta)$ is finite and uniquely determined by u. Thus, for $u \in V_{L,n}$ and $\varphi : K_{L,n} \to \mathbb{C} \cup \{\infty\}$ as above, we may associate to any polynomial $f := \sum a_{i_1 \ldots i_n} x_1^{i_1} \ldots x_n^{i_n} \in R_{L,n}[x_1, \ldots, x_n]$ the polynomial

$$f(u, x_1, \ldots x_n) := \sum a_{i_1 \ldots i_n}(u) x_1^{i_1} \ldots x_n^{i_n} := \sum \varphi(a_{i_1 \ldots i_n}) x_1^{i_1} \ldots x_n^{i_n},$$

which belongs to $\mathbb{C}[x_1, \ldots, x_n]$.

Since $\mu_{L,n} : \overline{W}_{L,n} \to V_{L,n}$ is a finite, bijective and birational morphism of affine varieties, we conclude that $R_{L,n}$ contains the coordinate ring of the affine variety $\overline{W}_{L,n}$. This implies that there exists a polynomial $f_{L,n} \in R_{L,n}[x_1, \ldots, x_n] \subset K[x_1, \ldots x_n]$ with the following property: for any $u \in V_{L,n}$, $x \in \mathbb{C}^n$ and $y \in \mathbb{C}$, the point (u, x, y) belongs to $B_{L,n}$ if and only if $f_{L,n}(u, x) = y$ holds.

A branching-free output representation of the extended sample point query associated to the constructible set $A_{L,n}$ can now easily be realized by any arithmetic circuit which first computes all monomial terms of the polynomial $f_{L,n}$ and finally sums them up.

Therefore, the given branching-parsimonious query evaluation procedure $\mathcal{P}$ produces on input consisting of the database D and the formula $\Phi_{L,n}(S,P)$ a branching-free representation of the extended sample point query associated to $A_{L,n}$. This branching-free representation is realized by an essentially division-free single-output arithmetic circuit $\beta_{L,n}$ with inputs $x_1,\ldots,x_n$ and scalars $\theta_1^{(L,n)},\ldots,\theta_{s_{L,n}}^{(L,n)}$ belonging to $R_{L,n}$ such that $\beta_{L,n}$ computes at its output the polynomial $f_{L,n}\in R_{L,n}[x_1,\ldots,x_n]$.

Let t and $\ell_1,\ldots,\ell_n$ be new variables and let us now consider the polynomial

$$g_n := t\prod_{1\le i\le n}(\ell_i+x_i)$$

defining the constructible object class

$$\Gamma_n := \{\tau\prod_{1\le i\le n}(\lambda_i+x_i)\ \mid\ \tau,\lambda_1,\ldots,\lambda_n\in\mathbb{C}\}$$

of n-variate complex polynomial functions. Observe that each element of Γ_n has nonscalar sequential time complexity at most n. Therefore the Zariski-closure $\overline{\Gamma}_n$ of the object class Γ_n is contained in $\overline{W}_{n,n}$.

Moreover, $\overline{\Gamma}_n$ is an absolutely irreducible, $\mathbb{Q}$-definable affine variety whose rational function field over $\mathbb{C}$ is denoted by L_n. Since $\mu_{n,n}:\overline{W}_{n,n}\to V_{n,n}$ is a finite morphism of irreducible affine varieties, we conclude that $C_n := \mu_{n,n}(\overline{\Gamma}_n)$ is an absolutely irreducible, $\mathbb{Q}$-definable closed affine subvariety of $V_{n,n}$.

The rational functions $\theta_1^{(n,n)},\ldots,\theta_{s_{n,n}}^{(n,n)}$ of the affine variety $V_{n,n}$ may be not defined on the subvariety C_n. Nevertheless, since they belong to the $\mathbb{C}$-algebra $R_{n,n}$, one verifies easily that there exist rational functions $\sigma_1^{(n)},\ldots,\sigma_{s_{n,n}}^{(n)}$ of the affine variety C_n satisfying the following condition: for any point $u\in C_n$ and any place $\psi: L_n\to\mathbb{C}\cup\{\infty\}$ whose evaluation ring extends the local ring of C_n at the point u, the values of ψ at $\sigma_1^{(n)},\ldots,\sigma_{s_{n,n}}^{(n)}$ are given by $\psi(\sigma_1^{(n)}) = \theta_1^{(n,n)}(u),\ldots,\psi(\sigma_{s_{n,n}}^{(n)}) = \theta_{s_{n,n}}^{(n,n)}(u)$ and therefore finite and uniquely determined by u.

This implies that we may consider the rational functions $\sigma_1^{(n)},\ldots,\sigma_{s_{n,n}}^{(n)}$ as polynomials in the variables t and $\ell_1,\ldots,\ell_n$ (i.e. as elements of $\mathbb{C}[t,\ell_1,\ldots,\ell_n]$) and that the specialized polynomials $\sigma_1^{(n)}(0,\ell_1,\ldots,\ell_n),\ldots,\sigma_{s_{n,n}}^{(n)}(0,\ell_1,\ldots,\ell_n)$ do not depend on the variables $\ell_1,\ldots,\ell_n$, i.e., they belong to $\mathbb{C}$.

Let $a_n := (\sigma_1^{(n)}(0,\ell_1,\ldots,\ell_n),\ldots,\sigma_{s_{n,n}}^{(n)}(0,\ell_1,\ldots,\ell_n))$ be the corresponding point of the affine space $\mathbb{C}^{s_{n,n}}$. Let now $\widetilde{\beta}_n$ be the division-free arithmetic circuit with scalars in $\mathbb{C}[t,\ell_1,\ldots,\ell_n]$ obtained by replacing in the circuit $\beta_{n,n}$ the scalars $\theta_1^{(n,n)},\ldots,\theta_{s_{n,n}}^{(n,n)}$ by the polynomials $\sigma_1^{(n)},\ldots,\sigma_{s_{n,n}}^{(n)}$.

One verifies easily that the circuit $\widetilde{\beta}_n$ computes the polynomial

$$g_n = t\prod_{1\le i\le n}(\ell_i+x_i) = \sum_{\substack{\delta_1,\ldots,\delta_n,\epsilon_1,\ldots,\epsilon_n\in\{0,1\}\\ \delta_1+\epsilon_1=1,\ldots,\delta_n+\epsilon_n=1}} t\ell_1^{\delta_1}\ldots\ell_n^{\delta_n}x_1^{\epsilon_1}\ldots x_n^{\epsilon_n}.$$

Let $v_1, \ldots, v_{s_{n,n}}$ be new variables. From the directed acyclic graph structure of $\widetilde{\beta}_n$ (or $\beta_{n,n}$) one deduces immediately that for each $(\delta_1, \ldots, \delta_n) \in \{0,1\}^n$ there exists a polynomial $Q^{(n)}_{(\delta_1,\ldots,\delta_n)} \in \mathbb{Q}[v_1, \ldots, v_{s_{n,n}}]$ satisfying the condition

$$Q^{(n)}_{(\delta_1,\ldots,\delta_n)}(\sigma^{(n)}_1, \ldots, \sigma^{(n)}_{s_{n,n}}) = t\ell_1^{\delta_1} \ldots \ell_n^{\delta_n}. \tag{1}$$

Let $Q_n : \mathbb{C}^{s_{n,n}} \to \mathbb{C}^{2^n}$ the polynomial map defined by $Q_n := (Q^{(n)}_{(\delta_1,\ldots,\delta_n)}; (\delta_1, \ldots, \delta_n) \in \{0,1\}^n)$ and let $\eta_n : \mathbb{C}^{s_{n,n}} \to \mathbb{C}^{2^n}$ be the $\mathbb{C}$-linear map defined by the derivative of Q_n at the point a_n of the affine space $\mathbb{C}^{s_{n,n}}$.

Observe that $Q_n(a_n)$ is the origin of $\mathbb{C}^{2^n}$.

By means of arguments of elementary analytic geometry we deduce now from (1) and the definition of the point a_n that the $\mathbb{C}$-linear map η_n is surjective. This implies $s_{n,n} \geq 2^n$.

Therefore the arithmetic circuit $\beta_{n,n}$, which represents the output produced by the procedure $\mathcal{P}$ on input consisting of the database D and the formula $\Phi_{n,n}$, contains at least 2^n scalars.

This implies that the nonscalar size of the circuit $\beta_{n,n}$ is at least $2^{\frac{n}{2}} - n - 1$. In conclusion, the procedure $\mathcal{P}$ requires $2^{\Omega(n)}$ sequential time in order to produce the output $\beta_{n,n}$ on input consisting of the data base D and the size $O(n^2)$ formula $\Phi_{n,n}$. □

The main outcome of Theorem 1 and its proof can be paraphrased as follows: *constraint database theory applied to quite natural computation tasks, as, e.g., branching-parsimonious interpolation of low complexity polynomials, leads necessarily to non-polynomial sequential time lower bounds.*

In view of the $\mathrm{P}_{\mathbb{R}} \neq \mathrm{NP}_{\mathbb{R}}$ conjecture in the algorithmic model of Blum–Shub–Smale over the real and complex numbers, it seems unlikely that this worst case complexity behavior can be improved substantially if we drop some or all of our previously introduced requirements on queries and their output representations. Nevertheless we wish to stress that these requirements constitute a fundamental technical ingredient for the argumentation in the proof of Theorem 1.

5 Conclusion and Future Research on the Complexity of Query Evaluation

In this paper, we have emphasized the importance of *data structures* and their effect on the complexity of quantifier elimination. We have also proposed a novel data model for constraint databases, consisting of *smooth cells accompanied by sample points,* as produced by the known most efficient elimination procedures.

However, the intrinsic inefficiency of quantifier-elimination procedures represents a bottle-neck for real-world implementations of constraint database systems. As we have argued, it is unlikely that constraint database systems that are based on general purpose quantifier-elimination algorithms will ever become efficient. Also, restriction to work with linear data, as in most existing constraint

database systems [28, Part IV], will also not lead to more efficiency. A promising direction is the study of a concept like the *system degree,* that has shown to be a fruitful notion for the complexity analysis of quantifier elimination in elementary geometry. In the context of query evaluation in constraint databases, the notion of system degree is still unsatisfactory since it is determined both by the query formula and the quantifier-free formulas describing the input database relations. It is a task for future constraint database research to develop a well-adapted complexity invariant in the spirit of the system degree in elimination theory.

Another direction of research is the study of query evaluation for first-order languages that capture certain genericity classes. For example, the first-order logic FO(between) has point variables rather than being based on real numbers and it captures the fragment of first-order logic over the reals that expresses queries that are invariant under affine transformations of the ambient space [19]. Although a more efficient complexity of query evaluation in this language cannot be expected, it is interesting to know whether languages such as FO(between) have quantifier elimination themselves (after an augmentation with suitable predicates).

Acknowledgments

The first author wishes to thank to Alejandro Vaisman, Pablo Solernó, Universidad de Buenos Aires, and especially to Lidia Quintero for many stimulating discussions supporting his start in the field of database theory.

The second author wishes to thank Joos Heintz and the members of his research group for their hospitality at the University of Buenos Aires during his visit supported by the project "Efficient Query Evaluation for Constraint Database Systems" of the FWO-Vlaanderen and SECyT, Argentina.

References

1. M. F. Atiyah and I. G. Macdonald. *Introduction to commutative algebra.* Addison-Wesley, 1969.
2. B. Bank, M. Giusti, J. Heintz, and G. M. Mbakop. Polar varieties, real equation solving and data structures: The hypersurface case. *Journal of Complexity,* 13:5–27, 1997. Best Paper Award Journal of Complexity 1997.
3. B. Bank, M. Giusti, J. Heintz, and G. M. Mbakop. Polar varieties and efficient real elimination. *Mathematische Zeitschrift,* 238:115–144, 2001.
4. B. Bank, M. Giusti, J. Heintz, and L. M. Pardo. Generalized polar varieties and an efficient real elimination procedure. *Submitted to Kibernetika,* 2003.
5. B. Bank, J. Guddat, D. Klatte, B. Kummer, and K. Tammer. *Non-Linear Parametric Optimization,* Birkhauser Verlag, Basel, 1983.
6. S. Basu, R. Pollack, and M.-F. Roy. On the combinatorial and algebraic complexity of quantifier elimination. *Journal of the ACM,* 43:1002–1045, 1996.
7. S. Berkowitz. On computing the determinant in small parallel time using a small number of processors. *Information Processing Letters,* 18:147–150, 1984.

8. L. Blum, F. Cucker, M. Shub, and S. Smale. *Complexity and real computation.* Springer-Verlag, New York, 1998.
9. J. Bochnak, M. Coste, and M.-F. Roy. *Real Algebraic Geometry,* volume 36 of *Ergebenisse der Mathematik und ihrer Grenzgebiete, Folge 3.* Springer-Verlag, 1998.
10. P. Bürgisser, M. Clausen, and M. A. Shokrollahi. *Algebraic Complexity Theory.* Springer-Verlag, 1997.
11. D. Castro, M. Giusti, J. Heintz, G. Matera, and L. M. Pardo. The hardness of polynomial solving. *Foundations of Computational Mathematics,* 3:347–420, 2003.
12. G. E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In *Automata Theory and Formal Languages,* volume 33 of *Lecture Notes in Computer Science,* pages 134–183. Springer-Verlag, 1975.
13. M. Giusti, K. Hägele, J. Heintz, J. E. Morais, J. L. Montaña, and L. M. Pardo. Lower bounds for diophantine approximation. *Journal of Pure and Applied Algebra,* 117&118:277–317, 1997.
14. M. Giusti and J. Heintz. Kronecker's smart, little black boxes. In R. DeVore, Iserles A., and E. Suli, editors, *Foundations of Computational Mathematics,* pages 69–104, Cambridge, 2001. Cambridge University Press.
15. M. Giusti, J. Heintz, J. E. Morais, J. Morgenstern, and L. M. Pardo. Straight–line programs in geometric elimination theory. *Journal of Pure and Applied Algebra,* 124:101–146, 1998.
16. M. Giusti, J. Heintz, J. E. Morais, and L. M. Pardo. Le rôle des structures des données dans les problèmes d'élimination. *Comptes Rendues de l'Académie des Sciences,* 325(Série I):1223–1228, 1997.
17. M. Giusti, G. Lecerf, and B. Salvy. A Gröbner free alternative for polynomial system solving. *Journal of Complexity,* 17(1):154–211, 2001.
18. D. Grigor'ev and N. N. (jr.) Vorobjov. Solving systems of polynomial inequalities in subexponential time. *Journal of Symbolic Computation,* 5:37–64, 1988.
19. M. Gyssens, J. Van den Bussche, and D. Van Gucht. Complete geometrical query languages. *Journal of Computer and System Sciences,* 58(3):483–511, 1999. A preliminary report appeared in the *Proceedings 16th ACM Symposium on Principles of Database Systems (PODS'97).*
20. J. Heintz, T. Krick, S. Puddu, J. Sabia, and A. Waissbein. Deformation techniques for efficient polynomial equation solving. *Journal of Complexity,* 16:70–109, 2000.
21. J. Heintz, G. Matera, and A. Weissbein. On the time–space complexity of geometric elimination procedures. *Applicable Algebra in Engineering, Communication and Computing,* 11(4):239–296, 2001.
22. J. Heintz and J. Morgenstern. On the intrinsic complexity of elimination theory. *Journal of Complexity,* 9:471–498, 1993.
23. J. Heintz, M.-F. Roy, and P. Solernó. Sur la complexité du principe de Tarski-Seidenberg. *Bulletin de la Société Mathématique de France,* 118:101–126, 1990.
24. G. Jeronimo, T. Krick, J. Sabia, and M. Sombra. The computational complexity of the Chow form. *Foundations of Computational Mathematics,* 4:41–117, 2004.
25. G. Jeronimo and J. Sabia. On the number of sets definable by polynomials. *Journal Algebra,* 227(2):633–644, 2000.
26. P. C. Kanellakis, G. M. Kuper, and P. Z. Revesz. Constraint query languages. *Journal of Computer and System Science,* 51(1):26–52, 1995. A preliminary report appeared in the *Proceedings 9th ACM Symposium on Principles of Database Systems (PODS'90).*
27. T. Krick, L. M. Pardo, and M. Sombra. Sharp estimates for the arithmetic Nullstellensatz. *Duke Mathematical Journal,* 109(3):521–598, 2001.

28. G. M. Kuper, J. Paredaens, and L. Libkin. *Constraint databases.* Springer-Verlag, 2000.
29. S. Lang. *Algebra.* Addison-Wesley Publishing Company, Reading, Massachusetts, 1969.
30. G. Lecerf. Quadratic Newton iterations for systems with multiplicity. *Foundations of Computational Mathematics,* 2:247–293, 2002.
31. J. Paredaens, J. Van den Bussche, and D. Van Gucht. Towards a theory of spatial database queries. In *Proceedings of the 13th ACM Symposium on Principles of Database Systems (PODS'94),* pages 279–288, New York, 1994. ACM Press.
32. J. Renegar. On the computational complexity and geometry of the first-order theory of the reals I, II, III. *Jornal of Symbolic Computation,* 13:255–352, 1992.
33. P. Revesz. *Introduction to Constraint Databases.* Springer-Verlag, 2002.
34. E. Schost. Computing parametric geometric resolutions. *Applicable Algebra in Engineering, Communication and Computing,* 13(5):349–393, 2003.
35. I. R. Shavarevich. *Basic Algebraic Geometry : Varieties in Projective Space.* Springer-Verlag, 1994.
36. J. von zur Gathen. Parallel linear algebra. In J. Reif, editor, *Synthesis of parallel algorithmns,* pages 573–617. Morgan-Kaufmann, 1993.

A New Shape Function Based Spatiotemporal Interpolation Method

Lixin Li[1], Youming Li[1], and Reinhard Piltner[2]

[1] Georgia Southern University, Statesboro, GA 30460-7997, USA
{lli,yming}@georgiasouthern.edu
[2] University of Nebraska-Lincoln, Lincoln, NE 68588-0526, USA
rpiltner@unlserve.unl.edu

Abstract. In this paper we propose a new spatiotemporal interpolation method for 3-D space and 1-D time geographic data. Similarly to the existing ST (space-time) product and the tetrahedral spatiotemporal interpolation methods, this new method is also based on shape functions. However, instead of only manipulating the time dimension as in the ST product and the tetrahedral methods, our new method combines 2-D shape functions in the (x,y) domain with the (z,t) domain shape functions. This method yields data that can be represented and queried in constraint databases and we discuss how to represent this new method in constraint databases. We also show certain unique properties of the new shape function based spatiotemporal interpolation method.

1 Introduction

Geographic Information System (GIS) [3, 4, 24, 39] applications increasingly require the use of spatiotemporal data, that is, data that combine both space and time [19]. For example, land-use change through time is a typical spatiotemporal data application. Future GIS systems need to efficiently manage spatiotemporal databases (STDBs) containing such data, which is usually beyond the capability of traditional relational databases. The study of the representation and the algorithmic methods to query and visualize spatiotemporal data in efficient ways is still a growing research area. There has been some promising progress in this area by using constraint databases [15, 17, 29, 37] as fundamentals for GIS systems. For example, MLPQ (Management Of Linear Programming Queries) [16, 30], is a GIS system based on linear constraint databases and is able to handle some spatiotemporal data.

GIS applications often require *spatiotemporal interpolation* of an input data set. Spatiotemporal interpolation requires the estimation of the unknown values at unsampled location-time pairs with a satisfying level of accuracy. For example, suppose that we know the recording of temperatures at different weather stations at different instances of time. Then spatiotemporal interpolation would estimate the temperature at unsampled locations and times.

Spatial interpolation is already frequently used in GIS. There are many *spatial interpolation* algorithms for spatial (2-D or 3-D) data sets. References [33]

B. Kuijpers and P. Revesz (Eds.): CDB 2004, LNCS 3074, pp. 25–39, 2004.

and [31] discuss in detail *inverse distance weighting* (IDW). Other methods useful for GIS interpolations are *kriging* [5], *splines* [10], *trend surfaces* [42], and *Fourier series* [13], Reference [18] offers a review and comparison of spatial interpolation methods.

Some effort has been devoted to the construction of spatiotemporal interpolations. Reference [11] considers the snapshots of the trajectory of moving points as a sample of points with time and position and uses linear interpolation between the snapshots to recover the full trajectory. It treats time as an independent variable. Reference [36] discusses the interpolation between snapshots of moving regions. Each moving region is represented as a set of polygons with polygonal holes whose vertices move linearly with time. The interpolation between two snapshots is based on a *sliced representation* that was introduced in reference [6]. Reference [2] also considers the interpolation of snapshots of moving regions by using sets of *parametric rectangles.*

Several researchers noticed that interpolation constraints can be stored in *constraint relations* and can make the evaluation of queries more feasible and efficient. For example, reference [31] represents the 2-D IDW spatial interpolation using polynomial constraint databases [14], references [8,9] investigate linear approximations of spatial databases in the plane that can be defined by polynomial constraint databases, and reference [32] represents the 2-D shape function based spatial interpolation using linear constraint databases. Representing interpolation in constraint databases is an advantage since there are many constraint database queries that are not expressible in current geographic information systems.

However, there are surprisingly few papers that consider the topic of spatiotemporal interpolation in GIS. One exception is [25], which utilizes kriging for spatiotemporal interpolation. Other exceptions are the more recent papers [20, 22, 21].

Based on shape functions [1, 41] from finite element methods (see. e.g. [40, 41, 27,26]), reference [20] discusses 2-D space & 1-D time spatiotemporal interpolation methods, tests the methods on an actual real estate database, and shows that the best method yields a spatiotemporal interpolation that estimates house prices (per square foot) with approximately 10 percent error. Reference [22] adopts shape functions for the spatiotemporal interpolation of both 2-D space & 1-D time and 3-D space & 1-D time data sets. New 4-D shape functions are developed and used for the spatiotemporal interpolation of 3-D space and 1-D time data sets. Using an actual real estate data set with house prices, [22] compares these methods with other spatiotemporal interpolation methods based on inverse distance weighting and kriging. Reference [21] gives the overview of three major types of GIS-oriented spatiotemporal databases: (1) point-based, (2) region-based, and (3) constraint-based, analyzes the relationship among these spatiotemporal databases, and shows how they can be translated into each other.

This paper is an extension for the work in [22] by introducing a new shape function based spatiotemporal interpolation method for 3-D space and 1-D time problems and representing this spatiotemporal method in constraint databases.

The rest of this paper is organized as follows. Section 2 gives a review on two existing shape functions based spatiotemporal interpolation methods for 3-D space and 1-D time problems. Section 3 describes a new shape function based spatiotemporal interpolation method for 3-D space and 1-D time problems. Section 4 discusses certain properties of the new interpolation method. Section 5 discusses how to represent this new shape-function based spatiotemporal interpolation method in constraint databases. Finally, in Section 6, we present a brief discussion of the future work.

2 Shape Function Based Spatiotemporal Interpolation Methods for 3-D Space and 1-D Time Problems

In this section, we give an overview about shape function based spatiotemporal interpolation methods for 3-D space and 1-D time problems. References [20,22] describe two general methods for spatiotemporal interpolations: (i) the *reduction method,* which treats time independently from the spatial dimensions; (ii) the *extension method,* which treats time as equivalent to a spatial dimension. [22] discusses shape function based reduction and extension methods to approach 3-D space and 1-D time spatiotemporal interpolation problems, and compares these methods with IDW (inverse distance weighting) and kriging based reduction and extension methods using a set of actual real estate data.

2.1 Reduction Approach

This shape function based reduction data approximation in 3-D space and 1-D time can be described in the following two steps: 3-D spatial interpolation by shape functions for tetrahedra and approximation in space and time.

Approximation in 3-D Space. Three-dimensional domains can be divided into finite number of simple sub-domains. For example, we can use tetrahedral or hexahedral sub-domains. Tetrahedral meshing is of particular interest. With a large number of tetrahedral elements, we can also approximate complicated 3-D objects. Several methods exist to generate automatic tetrahedral meshes, such as the 3-D Delaunay algorithm and some tetrahedral mesh improvement methods to avoid poorly-shaped tetrahedra. Examples are the tetrahedral mesh generation by Delaunay refinement [35] and tetrahedral mesh improvement using swapping and smoothing [7].

A linear approximation function for a 3-D tetrahedral element can be written in terms of four shape functions N_1, N_2, N_3, N_4 and the corner values w_1, w_2, w_3, w_4. In Figure 1, two tetrahedral elements, I and II, cover the whole domain considered.

In this example, the function in the whole domain is interpolated using five discrete values w_1, w_2, w_3, w_4, and w_5 at five locations in space. To obtain the function values inside the tetrahedral element I, we can use the four corner

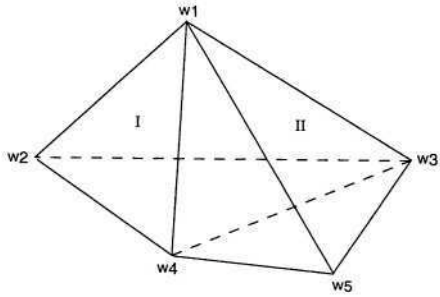

Fig. 1. Linear Interpolation in Space for Tetrahedral Elements.

values w_1, w_2, w_3 and w_4. Similarly, all function values for element II can be constructed using the corner values w_1, w_3, w_4 and w_5. The linear interpolation function for element I can be written as:

$$w(x,y,z) = N_1(x,y,z)w_1 + N_2(x,y,z)w_2 + N_3(x,y,z)w_3 + N_4(x,y,z)w_4$$

$$= [N_1\ N_2\ N_3\ N_4] \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix}, \tag{1}$$

where N_1, N_2 N_3 and N_4 are the following shape functions:

$$N_1(x,y,z) = \frac{a_1 + b_1x + c_1y + d_1z}{6\mathcal{V}}, N_2(x,y,z) = \frac{a_2 + b_2x + c_2y + d_2z}{6\mathcal{V}},$$
$$N_3(x,y,z) = \frac{a_3 + b_3x + c_3y + d_3z}{6\mathcal{V}}, N_4(x,y,z) = \frac{a_4 + b_4x + c_4y + d_4z}{6\mathcal{V}}. \tag{2}$$

The volume $\mathcal{V}$ of the tetrahedron used for the shape functions in (2) can be computed using the corner coordinates (x_i, y_i, z_i) $(i = 1,2,3,4)$ in the determinant of a 4×4 matrix according to

$$\mathcal{V} = \frac{1}{6}\ det \begin{bmatrix} 1 & x_1 & y_1 & z_1 \\ 1 & x_2 & y_2 & z_2 \\ 1 & x_3 & y_3 & z_3 \\ 1 & x_4 & y_4 & z_4 \end{bmatrix}. \tag{3}$$

a_i, b_i, c_i, and d_i $(i = 1,2,3,4)$ are some constants. By expanding the other relevant determinants into their cofactors, we have

$$a_1 = \ det \begin{bmatrix} x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \\ x_4 & y_4 & z_4 \end{bmatrix} \qquad b_1 = -det \begin{bmatrix} 1 & y_2 & z_2 \\ 1 & y_3 & z_3 \\ 1 & y_4 & z_4 \end{bmatrix}$$

$$c_1 = -det \begin{bmatrix} x_2 & 1 & z_2 \\ x_3 & 1 & z_3 \\ x_4 & 1 & z_4 \end{bmatrix} \qquad d_1 = -det \begin{bmatrix} x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \\ x_4 & y_4 & 1 \end{bmatrix}$$

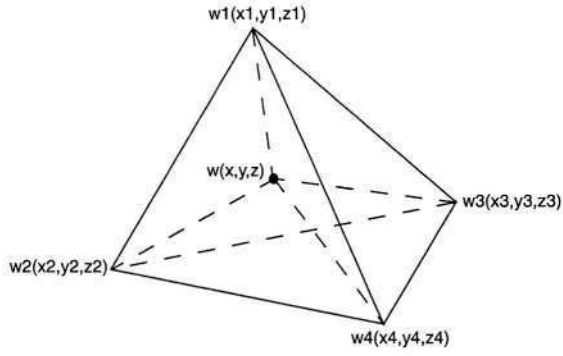

Fig. 2. Computing shape functions by volume divisions.

with the other constants defined by cyclic interchange of the subscripts in the order 4, 1, 2, 3 [40].

Alternatively, considering only the tetrahedral element I, the 3-D shape function (2) can also be expressed as follows:

$$N_1(x,y,z) = \frac{\mathcal{V}_1}{\mathcal{V}}, \quad N_2(x,y,z) = \frac{\mathcal{V}_2}{\mathcal{V}}, \quad N_3(x,y,z) = \frac{\mathcal{V}_3}{\mathcal{V}}, \quad N_4(x,y,z) = \frac{\mathcal{V}_4}{\mathcal{V}} \qquad (4)$$

$\mathcal{V}_1, \mathcal{V}_2, \mathcal{V}_3$ and $\mathcal{V}_4$ are the volumes of the four sub-tetrahedra $ww_2w_3w_4$, $w_1ww_3w_4$, $w_1w_2ww_4$, and $w_1w_2w_3w$, respectively, as shown in Figure 2; and $\mathcal{V}$ is the volume of the outside tetrahedron $w_1w_2w_3w_4$ which can be computed by equation (3). All the $\mathcal{V}_i$'s $(1 \leq i \leq 4)$ can also be computed similarly to equation (3) by using the appropriate coordinate values.

Approximation in 3-D Space and 1-D Time. Since in the reduction approach we model time independently, approximation in space and time can be implemented by combining a time shape function (5) with the space approximation function (1).

Assume the value at node i at time t_1 is w_{i1}, and at time t_2 the value is w_{i2}. The value at the node i at any time between t_1 and t_2 can be approximated using two 1-D time shape functions in the following way:

$$w_i(t) = \frac{t_2 - t}{t_2 - t_1} w_{i1} + \frac{t - t_1}{t_2 - t_1} w_{i2} \ . \qquad (5)$$

Using the example shown in Figure 1, the approximation function for any point constraint to the sub-domain I at any time between t_1 and t_2 can be expressed as follows:

$$\begin{aligned} w(x,y,z,t) = \quad & N_1(x,y,z) \left[\frac{t_2 - t}{t_2 - t_1} w_{11} + \frac{t - t_1}{t_2 - t_1} w_{12} \right] \\ & + N_2(x,y,z) \left[\frac{t_2 - t}{t_2 - t_1} w_{21} + \frac{t - t_1}{t_2 - t_1} w_{22} \right] \end{aligned}$$

$$+ N_3(x,y,z)\left[\frac{t_2-t}{t_2-t_1}w_{31}+\frac{t-t_1}{t_2-t_1}w_{32}\right]$$
$$+ N_4(x,y,z)\left[\frac{t_2-t}{t_2-t_1}w_{41}+\frac{t-t_1}{t_2-t_1}w_{42}\right] \tag{6}$$

$$= \frac{t_2-t}{t_2-t_1}\left[N_1(x,y,z)w_{11}+N_2(x,y,z)w_{21}+N_3(x,y,z)w_{31}+N_4(x,y,z)w_{41}\right]$$
$$+ \frac{t-t_1}{t_2-t_1}\left[N_1(x,y,z)w_{12}+N_2(x,y,z)w_{22}+N_3(x,y,z)w_{32}+N_4(x,y,z)w_{42}\right] .$$

Since the space shape functions (N_1, N_2, N_3 and N_4) and the time shape functions (5) are linear, the spatiotemporal approximation function (6) is bilinear.

2.2 Extension Approach

This method treats *time* as a regular fourth dimension. Linear 4-D shape functions are needed to solve this problem. In the engineering area, the highest number of dimensions of shape functions is three because there are no higher dimensional real objects. In reference [22], 4-D shape functions are developed.

The linear 4-D shape functions are based on 4-D Delaunay tesselation. The Delaunay tesselation in 4-D space is a special case of n-D space Delaunay tesselation when $n = 4$. The n-D Delaunay tesselation is defined as a space-filling aggregate of n-simplices [38]. Each Delaunay n-simplex can be represented by an (n+1)-tuple of indices to the data points. We can use Matlab to compute the n-D Delaunay tesselation by function *delaunayn*. $\mathbf{T}$ = *delaunayn*($\mathbf{X}$) computes a set of n-simplices such that no data points of $\mathbf{X}$ are contained in any n-D hyperspheres of the n-simplices. The set of n-simplices forms the n-D Delaunay tessellation. $\mathbf{X}$ is an $m \times n$ array representing m points in n-D space. $\mathbf{T}$ is an $s \times (n+1)$ array where s is the number of n-simplices after the n-D Delaunay tesselation. Each row of $\mathbf{T}$ contains the indices into $\mathbf{X}$ of the vertices of the corresponding n-simplex.

Two sets of different shape functions are developed in [22] using two different approaches. Since the first approach yields very long symbolic expressions whereas the second approach gives simple expressions, we only give the result for the second set of shape functions.

The linear interpolation function for a 4-simplex can be expressed as:

$$w(x,y,z,t) = \mathbf{N}(x,y,z,t)\mathbf{w}$$
$$= [N_1\ N_2\ N_3\ N_4\ N_5]\begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \end{bmatrix} \tag{7}$$

The shape functions N_i $(1 \leq i \leq 5)$ in (7) can be calculated as follows:

$$N_i = \hat{N}_i + \frac{m_i h}{detB} \ (1 \leq i \leq 4) \quad \text{and} \quad N_5 = \frac{h}{detB} \ , \tag{8}$$

where $\hat{N}_i$ $(1 \leq i \leq 4)$ denotes the 3-D shape functions (equations (2) or (4)), $m_i = \hat{N}_i(x_5, y_5, z_5)$ $(1 \leq i \leq 4)$, $detB = -m_1 t_1 - m_2 t_2 - m_3 t_3 - m_4 t_4 + t_5$ and $h = \hat{N}_1 t_1 + \hat{N}_2 t_2 + \hat{N}_3 t_3 + \hat{N}_4 t_4 - t$. This method can be generalized to derive shape functions of n dimension from shape functions of $n-1$ dimensions.

3 A New Shape Function Based Spatiotemporal Interpolation Method for 3-D Space and 1-D Time Problems

In the previous section, the reduction approach actually combines x,y,z information first and then multiply with the t information. The extension approach does the interpolation in x,y,z and t in one step. In this section, we propose a new approach to the Spatiotemporal interpolation for 3-D space and 1-D time problems, which interpolates in the (x, y) domain first and then multiply this (x, y) interpolation result with the interpolation in the (z, t) domain.

This approach can be described by two steps: (i) 2-D spatial interpolation by shape functions for triangles in the 2-D (x, y) domain, and (ii) approximation in 3-D space and 1-D time by combining (i) with the 2-D interpolation in the (z, t) domain.

3.1 Approximation in 2-D (x, y) Domain

When dealing with complex 2-D geometric domains, it is convenient to divide the total domain into a finite number of simple sub-domains which can have triangular or quadrilateral shapes. Mesh generation using triangular or quadrilateral domains is important in finite element discretization of engineering problems. For the generation of triangular meshes, quite successful algorithms have been developed. A popular method for the generation of triangular meshes is the "Delaunay Triangulation" [10, 28, 34]. Delaunay triangulation is related to the construction of the so called "Voronoi diagram", which is related to "Convex Hull" problems.

A linear approximation function for a triangular area can be written in terms of three shape functions N_1, N_2, N_3, and the corner values w_1, w_2, w_3. In Figure 3, two triangular finite elements, I and II, are combined to cover the whole domain considered.

In this example, the function in the whole domain is interpolated using four discrete values w_1, w_2, w_3, and w_4 at four locations. A particular feature of the chosen approximation method is that the function values inside the sub-domain I can be obtained by using only the three corner values w_1, w_2 and w_3, whereas all function values for the sub-domain II can be constructed using the corner

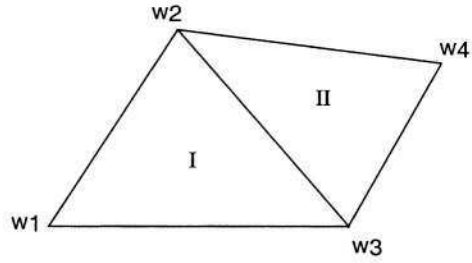

Fig. 3. Linear interpolation in 2-D space for triangular elements.

values w_2, w_3, and w_4. The linear interpolation function for the sub-domain of element I can be written as [22]

$$w(x,y) = N_1(x,y)w_1 + N_2(x,y)w_2 + N_3(x,y)w_3 = [N_1\ N_2\ N_3] \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} , \quad (9)$$

where N_1, N_2 and N_3 are the following shape functions:

$$\begin{aligned} N_1(x,y) &= \frac{[(x_2y_3 - x_3y_2) + x(y_2 - y_3) + y(x_3 - x_2)]}{2\mathcal{A}} \\ N_2(x,y) &= \frac{[(x_3y_1 - x_1y_3) + x(y_3 - y_1) + y(x_1 - x_3)]}{2\mathcal{A}} \\ N_3(x,y) &= \frac{[(x_1y_2 - x_2y_1) + x(y_1 - y_2) + y(x_2 - x_1)]}{2\mathcal{A}} . \end{aligned} \quad (10)$$

The area $\mathcal{A}$ used for the shape functions in equation (10) can be computed using the corner coordinates (x_i, y_i) $(i = 1, 2, 3)$ in the determinant of a 3×3 matrix according to

$$\mathcal{A} = \frac{1}{2}\ det \begin{bmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{bmatrix} . \quad (11)$$

For the sub-domain II, the local approximation takes a similar form as the expression (9): we just have to replace the corner values w_1, w_2 and w_3 with the new values w_2, w_3 and w_4.

It should be noted that for every sub-domain, a local approximation function similar to expression (9) is used. Each local approximation function is constrained to the local triangular sub-domain. For example, the function w of equation (9) is valid only for sub-domain I.

Alternatively, considering only sub-domain I, the 2-D shape function (10) can also be expressed as follows [32]:

$$N_1(x,y) = \frac{\mathcal{A}_1}{\mathcal{A}}, \quad N_2(x,y) = \frac{\mathcal{A}_2}{\mathcal{A}}, \quad N_3(x,y) = \frac{\mathcal{A}_3}{\mathcal{A}} . \quad (12)$$

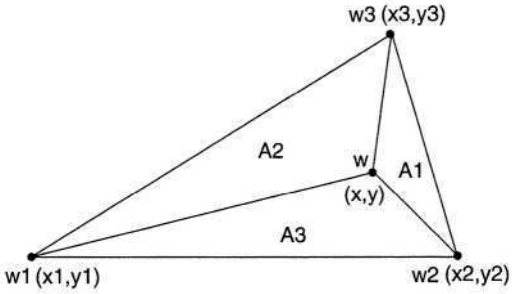

Fig. 4. Computing 2-D shape functions by area divisions.

$\mathcal{A}_1$, $\mathcal{A}_2$ and $\mathcal{A}_3$ are the three sub-triangle areas of sub-domain I as shown in Figure 4, and $\mathcal{A}$ is the area of the outside triangle $w_1w_2w_3$ which can be computed by equation (11). All the $\mathcal{A}_i$'s $(1 \leq i \leq 3)$ can also be computed in the similar way as equation (11) by using the appropriate coordinate values.

3.2 Approximation in 3-D Space and 1-D Time by Combining Approximations in (x, y) Domain and in (z, t) Domain

At this second step, we want to combine 2-D shape functions in the (z, t) domain with the (x, y) approximation function (9). First of all, we need to generate a triangular mesh in the (z, t) domain. Assume the value at node i in the (x, y) domain is located inside the triangle in the (z, t) domain with vertices $i1$, $i2$ and $i3$. The value at the node i in the (x, y) domain at any (z, t) pair that fits inside the triangle with vertices $i1$, $i2$ and $i3$ can be approximated using a 2-D shape function in the following way:

$$w_i(z,t) = \widehat{N}_{i1}(z,t)w_{i1} + \widehat{N}_{i2}(z,t)w_{i2} + \widehat{N}_{i3}(z,t)w_{i3} \ . \tag{13}$$

Here $\hat{N}_{i1}$, $\hat{N}_{i2}$ and $\hat{N}_{i3}$ are the following shape functions:

$$\begin{aligned}
\widehat{N}_{i1}(z,t) &= \frac{[(z_{i2}t_{i3} - z_{i3}t_{i2}) + z(t_{i2} - t_{i3}) + t(z_{i3} - z_{i2})]}{2\hat{\mathcal{A}}} \\
\widehat{N}_{i2}(z,t) &= \frac{[(z_{i3}t_{i1} - z_{i1}t_{i3}) + z(t_{i3} - t_{i1}) + t(z_{i1} - z_{i3})]}{2\hat{\mathcal{A}}} \\
\widehat{N}_{i3}(z,t) &= \frac{[(z_{i1}t_{i2} - z_{i2}t_{i1}) + z(t_{i1} - t_{i2}) + t(z_{i2} - z_{i1})]}{2\hat{\mathcal{A}}} \ .
\end{aligned} \tag{14}$$

The area $\hat{\mathcal{A}}$ used for the shape functions in equation (14) can be computed using the corner coordinates (z_{ik}, t_{ik}) $(k = 1, 2, 3)$ in the determinant of a 3×3 matrix according to

$$\hat{\mathcal{A}} = \frac{1}{2} \ det \begin{bmatrix} 1 & z_{i1} & t_{i1} \\ 1 & z_{i2} & t_{i2} \\ 1 & z_{i3} & t_{i3} \end{bmatrix} \ . \tag{15}$$

Using the example shown in Figure 3, the linear approximation function for any point constraint to the sub-domain I at any (z, t) pair that fits inside the (z, t) domain triangle with vertices $i1$, $i2$ and $i3$ can be expressed as follows:

$$w(x,y,z,t) = N_1(x,y)w_1(z,t) + N_2(x,y)w_2(z,t) + N_3(x,y)w_3(z,t). \quad (16)$$

Using nine nodal values we get

$$\begin{aligned} w(x,y,z,t) = {} & N_1(x,y)\widehat{N}_{11}(z,t)w_{11} + N_1(x,y)\widehat{N}_{12}(z,t)w_{12} + N_1(x,y)\widehat{N}_{13}(z,t)w_{13} \\ & + N_2(x,y)\widehat{N}_{21}(z,t)w_{21} + N_2(x,y)\widehat{N}_{22}(z,t)w_{22} + N_2(x,y)\widehat{N}_{23}(z,t)w_{23} \\ & + N_3(x,y)\widehat{N}_{31}(z,t)w_{31} + N_3(x,y)\widehat{N}_{32}(z,t)w_{32} + N_3(x,y)\widehat{N}_{33}(z,t)w_{33}\ , \end{aligned}$$

where N_i's $(1 \le i \le 3)$ can be obtained from equations (10).

Since the shape functions in the (x, y) domain (N_1, N_2 and N_3) and the shape functions in the (z, t) domain ($\hat{N}_{i1}$, $\hat{N}_{i2}$ and $\hat{N}_{i3}$) are linear, the spatiotemporal approximation function (16) involves the bilinear terms xz, yz, xt, and yt.

4 Properties of the New Shape Function Based Spatiotemporal Interpolation Method

The new shape function based spatiotemporal interpolation enjoys the property of invariance under not only coordinating scaling, (see e.g., [22] for the property for the 3-D space/1-D time reduction method.) but also certain types of linear transformations of the coordinate system. This property of invariance under linear transformations allows data to be manipulated under scaling, rotation or even shearing, according to actual convenience. In this section, we demonstrate the invariance property for our new interpolation method, as well as the reduction interpolation method for 3-D space/1-D time. Then we use the property to make certain initial theoretical observations on the accuracy of the two interpolation methods; more complete studies will be done in the future.

We first consider the interpolation for the case of 3-D space/1-D time; the case for the new method can be similarly derived. Let S be a 3×3 matrix with $\det(S) > 0$. For a point $w = (x, y, z)$ we denote by $w' = (x', y', z')$ the image of w under S: $w' = Sw^T$. The shape functions N_i for a 3-D tetrahedron with corners $w_i = (x_i, y_i, z_i)$ $(i = 1, \ldots, 4)$ can be represented as $N_i(x, y, z) = \mathcal{V}_i/\mathcal{V}$, where $\mathcal{V}_1, \mathcal{V}_2, \mathcal{V}_3, \mathcal{V}_4$ are the volumes of the four tetrahedra $T_1 = ww_2w_3w_4, T_2 = w_1ww_3w_4, T_3 = w_1w_2ww_4$ and $T_4 = w_1w_2w_3w$, respectively, and $\mathcal{V}$ is the volume of the tetrahedron $w_1w_2w_3w_4$. Let $T_i' = S(T_i)$, $T' = S(T)$ and $\mathcal{V}_i', \mathcal{V}'$ be the volumes of T_i' and T', respectively. Let $dW = dxdydz$ and $dW' = dx'dy'dz'$. Then $dW' = \det(S)\, dW$. Thus the shape functions $N_i'(x', y', z')$ for the tetrahedron $w_1'w_2'w_3'w_4'$ are

$$N_i' = \frac{\mathcal{V}_i'}{\mathcal{V}'} = \frac{\int_{T_i'} dW'}{\int_{T'} dW'} = \frac{\int_{T_i} \det(S)\, dW}{\int_{T} \det(S)\, dW} = \frac{\mathcal{V}_i}{\mathcal{V}} = N_i.$$

Moreover, if we employ scaling with respect to time t, then the shape function for the time is also invariant. Thus using the following matrix to transform the original data to new data will keep the shape functions, and thus the interpolation invariant:

$$\begin{bmatrix} S & 0 \\ 0 & a \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & 0 \\ a_{21} & a_{22} & a_{23} & 0 \\ a_{31} & a_{32} & a_{33} & 0 \\ 0 & 0 & 0 & a \end{bmatrix} \quad \text{with } \det(S) > 0 \text{ and } a > 0.$$

Now we consider invariance for our new interpolation method. The argument is the same as above, and thus details are omitted. The desired matrix is of the form

$$\begin{bmatrix} A & 0 \\ 0 & B \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & 0 & 0 \\ a_{21} & a_{22} & 0 & 0 \\ 0 & 0 & b_{11} & b_{12} \\ 0 & 0 & b_{21} & b_{22} \end{bmatrix} \quad \text{with } \det(A) > 0 \text{ and } \det(B) > 0.$$

The invariance property is convenient in certain situations. For instance, let us consider the problem of studying accuracy of our new interpolation method. We can reduce the problem to a problem with coordinates in the four-dimensional cube $[0,1]^4$ by simply performing scaling along each variable. This invariance property can also be used to improve numerical stability of triangulation. For instance, a triangle with small angles can be rescaled to one with regular shape. Here we restrict our attention to a few facts without details. We plan to report more complete theoretical results backed by real world examples in the future.

For the purpose of preciseness, we assume the data will have certain properties of continuity and smoothness, specified when needed. Thus under this theoretical setting, an interpolation method is used which utilizes piece-wise linear functions to approximate a function $w = f(x,y,z,t)$. Notice that our method is a special form of tensor product algorithms (see [23]). Let U be the (x,y) interpolation, and V the (z,t) interpolation. Then our interpolation can be represented as $U \otimes V$. If we know the errors of U and V, what is the error of $U \otimes V$? A sharp estimate of the error is given in [12]. The concept of "error" and the formula of the estimate account at least another section, and therefore will not be given here. Instead, we give the following description.

To ensure that U and V are convergent, smoothness must be assumed for f over $[0,1]^d$:

$$\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}, \frac{\partial f}{\partial t}, \frac{\partial^2 f}{\partial x \partial y}, \frac{\partial^2 f}{\partial z \partial t} \text{ exist, and are bounded in the } L_p \text{ norm.}$$

Intuitively, this smoothness is the same as saying that f has small "fluctuations" in a small neighborhood of any point (x,y,z,t) along (x,y) plane and (z,t) plane, respectively. To ensure that $U \otimes V$ is convergent, we need to assume the existence of the partial derivatives of the type $(|2|,|2|)$ (see [23]), with bounded

L_p norms. We point out that this type of derivatives does not include all the partial derivatives of order less than or equal to d, where $d = 3$ or $d = 4$.

As a conclusion of the section, we stress that for the 3-D space/1-D time reduction interpolation, the estimate of convergence rate has to involve all the third partial derivatives on x, y, z to achieve the same order of accuracy for U. For the 4-D extension method, all the fourth partial derivatives must be involved in the estimate for the same purpose. These requirements on other methods make our new method more convenient, at least theoretically.

5 Representing the New Shape Function Based Spatiotemporal Interpolation Method in Constraint Databases

In this section, we discuss how to represent and store our new shape function based Spatiotemporal interpolation in constraint databases. Assume we have the following input relations after Delaunay triangulations in the (x, y) and (z, t) domains:

1. $Point\_Value(x_1, y_1, z_1, t_1, w_1)$ stores the nodal value w_1 for the 4-D sample point (x_1, y_1, z_1, t_1).
2. $TriangleXY(x_1, y_1, x_2, y_2, x_3, y_3, x, y)$ indicates that in the (x, y) domain the point (x, y) is inside the triangle with three corner vertices (x_1, y_1), (x_2, y_2) and (x_3, y_3).
3. $TriangleZT(x_0, y_0, z_1, t_1, z_2, t_2, z_3, t_3, z, t)$ indicates that in the (z, t) domain the point (z, t) is inside the triangle with three corner vertices (z_1, t_1), (z_2, t_2) and (z_3, t_3). The first two attributes x_0 and y_0 indicate that this (z, t) domain triangle corresponds to the node x_0, y_0 in the (x, y) domain. That is, the following points are in the *Point_Value* relation: (x_0, y_0, z_1, t_1), (x_0, y_0, z_2, t_2), and (x_0, y_0, z_3, t_3).

Based on equations 13, 16 and 12, the value w of any point (x, y, z, t) can be represented by the following polynomial constraint tuple:

$$
\begin{aligned}
FourD\_Approx(x, y, z, t, w) :- & \; TriangleXY(x_1, y_1, x_2, y_2, x_3, y_3, x, y) \ , \\
& \; TwoD\_Approx(x_1, y_1, z, t, w_1) \ , \\
& \; TwoD\_Approx(x_2, y_2, z, t, w_2) \ , \\
& \; TwoD\_Approx(x_3, y_3, z, t, w_3) \ , \\
& \; Area(x_1, y_1, x_2, y_2, x_3, y_3, a) \ , \\
& \; Area(x, y, x_2, y_2, x_3, y_3, a_1) \ , \\
& \; Area(x_1, y_1, x, y, x_3, y_3, a_2) \ , \\
& \; Area(x_1, y_1, x_2, y_2, x, y, a_3) \ , \\
& \; wa = w_1 a_1 + w_2 a_2 + w_3 a_3 \ .
\end{aligned}
\tag{17}
$$

$$
\begin{aligned}
TwoD\_Approx(x_0, y_0, z, t, w) :- & \; TriangleZT(x_0, y_0, z_1, t_1, z_2, t_2, , z_3, t_3, z, t) \; , \\
& \; Point\_Value(x_0, y_0, z_1, t_1, w_1) \; , \\
& \; Point\_Value(x_0, y_0, z_2, t_2, w_2) \; , \\
& \; Point\_Value(x_0, y_0, z_3, t_3, w_3) \; , \\
& \; Area(z_1, t_1, z_2, t_2, z_3, t_3, a) \; , \\
& \; Area(z, t, z_2, t_2, z_3, t_3, a_1) \; , \\
& \; Area(z_1, t_1, z, t, z_3, t_3, a_2) \; , \\
& \; Area(z_1, t_1, z_2, t_2, z, t, a_3) \; , \\
& \; wa = w_1 a_1 + w_2 a_2 + w_3 a_3 \; .
\end{aligned}
\tag{18}
$$

where the $Area(u_1, v_1, u_2, v_2, u_3, v_3, a)$ relation calculates the area of the triangle with vertices (u_1, v_1), (u_2, v_2) and (u_3, v_3). Each of the (u, v) pair can represent either a point in the (x, y) domain or in the (z, t) domain.

$$
Area(u_1, v_1, u_2, v_2, u_3, v_3, a) : -a = \frac{\mid (u_1 - u_3)(v_2 - v_3) - (v_1 - v_3)(u_2 - u_3) \mid}{2} \; .
$$

6 Discussion and Future Work

In this paper we discuss a new shape function based spatiotemporal interpolation method for 3-D space and 1-D time problems. There are interesting areas for future work, which could include for example:

1. Use some 3-D space and 1-D time GIS data and test the new method in numericalexperiments.
2. Compare the new method with other existing methods, namely the reduction method and extension method based on the same data sets.
3. Explore theoretical aspects to compare the different methods.

References

1. G. R. Buchanan. *Finite Element Analysis.* McGraw-Hill, New York, 1995.
2. M. Cai, D. Keshwani, and P. Revesz. Parametric rectangles: A model for querying and animating spatiotemporal databases. In *Proc. 7th International Conference on Extending Database Technology,* volume 1777 of *Lecture Notes in Computer Science,* pages 430–444. Springer-Verlag, 2000.
3. Kang-tsung Chang. *Introduction to Geographic Information Systems.* McGraw-Hill, New York, 2nd edition, 2004.
4. M. N. Demers. *Fundamentals of Geographic Information Systems.* John Wiley & Sons, New York, 2nd edition, 2000.
5. C. V. Deutsch and A. G. Journel. *GSLIB: Geostatistical Software Library and User's Guide.* Oxford University Press, New York, 2nd edition, 1998.
6. L. Forlizzi, R. H. Güting, E. Nardelli, and M. Schneider. A data model and data structure for moving object databases. In *Proc. ACM SIGMOD International Conference on Management of Data,* pages 319–330, 2000.

7. L. A. Freitag and C. O. Gooch. Tetrahedral mesh improvement using swapping and smoothing. *International Journal for Numerical Methods in Engineering,* 40:3979–4002, 1997.
8. F. Geerts and B. Kuijpers. Deciding termination of query evaluation in transitive-closure logics for constraint databases. In *Proceedings of the 19th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems,* pages 126–135. ACM Press, 2000.
9. F. Geerts and B. Kuijpers. Linear approximation of planar spatial databases using transitive-closure logic. In *Proc. 9th International Conference on Database Theory,* volume 2572 of *Lecture Notes in Computer Science,* pages 190–206. Springer-Verlag, 2003.
10. J. E. Goodman and J. O'Rourke, editors. *Handbook of Discrete and Computational Geometry.* CRC Press, Boca Raton, New York, 1997.
11. S. Grumbach, P. Rigaux, and L. Segoufin. Manipulating interpolated data is easier than you thought. In *Proc. IEEE International Conference on Very Large Databases,* pages 156–165, 2000.
12. F. Hang and Y. Li. On applicability of sparse grid algorithms. *Numerical Algorithms,* submitted.
13. J. W. Harbaugh and F. W. Preston. *Fourier Analysis in Geology,* pages 218–238. Prentice-Hall, Englewood Cliffs, 1968.
14. J. Jaffar, S. Michaylov, P. J. Stuckey, and R. H. Yap. The CLP(R) language and system. *ACM Transactions on Programming Languages and Systems,* 14(3):339–395, 1992.
15. P. C. Kanellakis, G. M. Kuper, and P. Revesz. Constraint query languages. *Journal of Computer and System Sciences,* 51(1):26–52, 1995.
16. P. Kanjamala, P. Revesz, and Y. Wang. MLPQ/GIS: A GIS using linear constraint databases. In C. S. R. Prabhu, editor, *Proc. 9th COMAD International Conference on Management of Data,* pages 389–393, 1998.
17. G. M. Kuper, L. Libkin, and J. Paredaens, editors. *Constraint Databases.* Springer-Verlag, 2000.
18. N. S. Lam. Spatial interpolation methods: A review. *The American Cartographer,* 10(2):129–149, 1983.
19. G. Langran. *Time in Geographic Information Systems.* Taylor and Francis, London, 1992.
20. L. Li and P. Revesz. A comparison of spatio-temporal interpolation methods. In M. Egenhofer and D. Mark, editors, *Proc. of the Second International Conference on GIScience 2002,* volume 2478 of *Lecture Notes in Computer Science,* pages 145–160. Springer, 2002.
21. L. Li and P. Revesz. The relationship among GIS-oriented spatiotemporal databases. In *Proc. of the Third National Conference on Digital Government Research,* Boston, 2003.
22. L. Li and P. Revesz. Interpolation methods for spatio-temporal geographic data. *Journal of Computers, Environment and Urban Systems,* 2003 in press.
23. Y. Li. Applicability of Smolyak's algorithms to certain Banach spaces of multivariate functions. *Journal of Complexity,* 18(2):792–814, 2002.
24. P. A. Longley, M. F. Goodchild, D. J. Maguire, and D. W. Rhind. *Geographic Information Systems and Science.* John Wiley, Chichester, 2001.
25. E. J. Miller. Towards a 4D GIS: Four-dimensional interpolation utilizing kriging. In Z. Kemp, editor, *Innovations in GIS 4: Selected Papers from the Fourth National Conference on GIS Research U.K, Ch. 13,* pages 181–197, London, 1997. Taylor & Francis.

26. R. Piltner. Low order plate bending elements with enhanced strains. *Computers & Structures,* 80:849–856, 2002.
27. R. Piltner and R.L. Taylor. Triangular finite elements with rotational degrees of freedom and enhanced strain modes. *Computers & Structures,* 75:361–368, 2000.
28. F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction.* Springer-Verlag, 1985.
29. P. Revesz. *Introduction to Constraint Databases.* Springer, New York, 2002.
30. P. Revesz, R. Chen, P. Kanjamala, Y. Li, Y. Liu, and Y. Wang. The MLPQ/GIS constraint database system. In *Proc. ACM SIGMOD International Conference on Management of Data,* 2000.
31. P. Revesz and L. Li. Constraint-based visualization of spatial interpolation data. In *Proc. of the Sixth International Conference on Information Visualization,* pages 563–569, London, England, 2002. IEEE Press.
32. P. Revesz and L. Li. Representation and querying of interpolation data in constraint databases. In *Proc. of the Second National Conference on Digital Government Research,* pages 225–228, Los Angeles, California, 2002.
33. D. Shepard. A two-dimensional interpolation function for irregularly spaced data. In *Proc. 23nd National Conference ACM,* pages 517–524. ACM, 1968.
34. J. R. Shewchuk. Triangle: Engineering a 2D quality mesh generator and delaunay triangulator. In *Proc. First Workshop on Applied Computational Geometry,* pages 124–133, Philadelphia, Pennsylvania, 1996.
35. J. R. Shewchuk. Tetrahedral mesh generation by delaunay refinement. In *Proc. 14th Annual ACM Symposium on Computational Geometry,* pages 86–95, Minneapolis, Minnesota, 1998.
36. E. Tossebro and R. H. Güting. Creating representation for continuously moving regions from observations. In *Proc. 7th International Symposium on Spatial and Temporal Databases,* pages 321–344, Redondo Beach, CA, 2001.
37. J. Van den Bussche. Constraint databases: A tutorial introduction. *SIGMOD Record,* 29(3):44–51, 2000.
38. D. F. Watson. Computing the n-dimensional delaunay tesselation with application to voronoi polytopes. *The Computer Journal,* 24(2):167–172, 1981.
39. M. F. Worboys. *GIS: A Computing Perspective.* Taylor & Francis, 1995.
40. O. C. Zienkiewics and R. L. Taylor. *Finite Element Method, Vol. 1, The Basic Formulation and Linear Problems.* McGraw-Hill, 1989.
41. O. C. Zienkiewics and R. L. Taylor. *Finite Element Method, Vol. 1, The Basis.* Butterworth Heinemann, London, 2000.
42. E. G. Zurflueh. Applications of two-dimensional linear wavelength filtering. *Geophysics,* 32:1015–1035, 1967.

Moving Objects and Their Equations of Motion

Floris Geerts

Helsinki Institute for Information Technology
Basic Research Unit, Department of Computer Science
University of Helsinki, Finland
floris.geerts@cs.helsinki.fi

Abstract. Moving objects are currently represented in databases by means of an explicit representation of their trajectory. However from a physical point of view, or more specifically according to Newton's second law of motion, a moving object is fully described by its equation of motion. We introduce a new data model for moving objects in which a trajectory is represented by a differential equation. A similar approach is taken in computer animation where this is known as physically based modeling. We give a query language for our data model and use techniques from physically based modeling to evaluate queries in this language.

1 Introduction

Q: What's a moving object?
A: An equation of motion.

Moving objects pose many challenges for data management systems. These challenges include the modeling and representation of moving objects, query language design, indexing techniques and query optimization. In this paper we focus on the modeling of moving objects and the design of query languages.

Existing models involve temporal logic, abstract data types[7,9] and linear constraints [15,21,6]. In all these models the trajectories of the moving objects are explicitly stored in the database. Here, we take an alternative approach in which the sometimes complicated trajectories are stored in the form of preferably simpler equations of motions. This is somehow analogous to the constraint database approach [11,12,19] in which an infinite collection of points is stored in the form of geometric equations and inequalities.

Equations of motions are a natural way of describing moving objects. Many important motions have been described by equations since Newton published his Laws of Physics in 1687 [17]. The physics and mathematics of the equations of motion are by now well understood for large classes of moving objects like rigid bodies, deformable bodies, and so on [8,2]. In this paper we will only consider moving point objects.

The approach of working with equations instead of trajectories is very similar to what is known as physically based modeling in computer graphics [4], There,

B. Kuijpers and P. Revesz (Eds.): CDB 2004, LNCS 3074, pp. 40–51, 2004.

moving graphical objects are defined in terms of their equations of motions and when needed, the trajectories are computed using numerical integration techniques.

Our data model requires a new query language and evaluation method. Fortunately, the generalized distance based query language proposed by Mokthar et al. [15] can be adapted to our setting. Using techniques from physically based modeling we show how these queries can be evaluated on moving object database in which only equations of motions are represented.

It is surprising that the approach proposed in this paper has not been considered before in the database community. As described in the recent survey of Agarwal et al. [1], the area of physically based modeling might have many more techniques which can be useful for the development of moving object databases and vice versa. They also point out other interesting cross-disciplinary aspects related to moving objects. However, in this paper we only explore physically based modeling from a database point of view and regard the exploration of other connections as future work.

Organization of the paper: In Section 2 we briefly describe concepts related to the physics of motion. The data model for moving objects is defined in Section 3. We then describe the physical validity of moving object databases in Section 4. The query language is defined in Section 5 and its evaluation is described in Section 6. We conclude the paper in Section 7.

2 Physics and Simulation

We start with a brief summary of concepts needed to describe the physics of motion. For a detailed description we refer to standard physics textbooks [8,2] and the excellent SIGGRAPH tutorial of Baraff and Witkin [4]. In this paper we only consider point particles. Extensions to e.g. rigid body motions only require more physics and formulas and would deviate us too much from the aim of this paper.

Let $\boldsymbol{x}(t) = (x_1(t), \ldots, x_n(t))$ denote the location of point particle in $\mathbb{R}^n$ at time t. The velocity of the particle at time t is then given by $\boldsymbol{v}(t) = \dot{\boldsymbol{x}}(t) = \frac{d}{dt}\boldsymbol{x}(t)$. By Newton's Second Law, the motion is now fully described once we know the force $\boldsymbol{F}(t)$ acting on the particle (this force can be gravity, wind, spring forces, etc). Indeed, the motion is described by the unique solution of the (differential) equation of motion

$$\boldsymbol{F}(t) = m\frac{d^2}{dt^2}\boldsymbol{x}(t), \quad \boldsymbol{x}(t_0) = \boldsymbol{x}_0, \boldsymbol{v}(t_0) = \boldsymbol{v}_0 \tag{1}$$

in which m denotes the mass of particle and $\boldsymbol{x}(t_0) = \boldsymbol{x}_0$ and $\boldsymbol{v}(t_0) = \boldsymbol{v}_0$ are initial conditions.

Example 1. Consider a point particle moving only along the x-axis according to the equation of motion $-\omega^2 x(t) = \frac{d^2}{dt^2}x(t)$ where $\omega \in \mathbb{R}$. Solving this equation results in $x(t) = A\sin(\omega t + \varphi)$, where A and φ are real constants determined by the initial conditions. This motion is known as the harmonic oscillator. □

In the above example, the equation of motion can be solved analytically. However, this is not always the case and one often has to rely on numerical integration techniques to obtain some information about the solution [10]. Commonly used methods include Euler's method and higher-order (adaptive) Runge-Kutta methods. Most methods take first-order differential equations as input, meaning that only partial derivatives of the first order may occur in the equation. However, this forms no restriction since higher-order differential equations can always be transformed into first-order ones by adding variables: E.g., the second-order equation (1) is equivalent to the first-order equation

$$\frac{d}{dt}\boldsymbol{X}(t) = \frac{d}{dt}\begin{pmatrix}\boldsymbol{x}(t)\\ \boldsymbol{v}(t)\end{pmatrix} = \begin{pmatrix}\boldsymbol{v}(t)\\ \boldsymbol{F}(t)/m\end{pmatrix}. \tag{2}$$

When equation (2) together with an initial condition and an initial and final point in time t_0 and t_1 is fed to these integration methods, the output is a continuous piecewise linear curve $[t_0, t_1] \to \mathbb{R}^{2n}$ which approximates the real solution to the differential equations. The real solution of a differential equation is also called an *integral curve* and is uniquely determined by the equation and the initial conditions. Moreover, by Picard's existence theorem this integral curve always exists under some mild condition on the formulas defining the equations (the so-called Lipschitz condition). We will assume that these conditions are always satisfied.

3 A Model for Moving Objects

In this section we introduce a data model for moving objects. Unlike earlier models for moving objects that use ADTs [7,9] or linear constraints [15,21,6], we do not store the complete trajectories of the moving objects, but store the equations of motion to which the trajectories are the solution.

We will represent geometric objects by means of linear constraints. A *linear constraint* over variables $x_1, \ldots, x_n$ has the following general form: $\sum_{i=1}^{n} a_i x_i \theta a_o$, where $a_0, a_1, \ldots, a_n$ are integer numbers and θ is an order predicate $(=, <, \leq, >, \geq)$. Constraints are interpreted over the real numbers. We use a vector $\boldsymbol{x} = (x_1, \ldots, x_n)$ to denote a point in space. On top of the ordinary variables we also have the time variable t.

The position of a moving point can be modeled by a function of time, represented by the real line $\mathbb{R}$, to the n-dimensional space $\mathbb{R}^n$. A function from $\mathbb{R}$ to $\mathbb{R}^n$ is *linear* if it has the form $\boldsymbol{x} = \boldsymbol{a}t + \boldsymbol{b}$ where $\boldsymbol{a}, \boldsymbol{b}$ are vectors in $\mathbb{R}^n$; A function is *piecewise linear* if it consists of a finite number of linear pieces, i.e., if it has the form

$$\boldsymbol{x} = \begin{cases} \boldsymbol{a}_1 t + \boldsymbol{b}_1 & \text{if } t_0^{(1)} \leq t \leq t_1^{(1)} \\ \quad\vdots & \quad\vdots \\ \boldsymbol{a}_k t + \boldsymbol{b}_k & \text{if } t_0^{(k)} \leq t \leq t_1^{(k)}, \end{cases}$$

where $t_1^{(i)} \leq t_0^{(i+1)}$ for all $i = 1, \ldots, k$.

Definition 1. A *trajectory* is a piecewise linear function from $\mathbb{R}$ to $\mathbb{R}^n$. Let $\mathcal{T}$ be the set of all trajectories. □

We also introduce the *derivative variables* $\dot{x}_1, \ldots, \dot{x}_n$. Although we will consider them as an ordinary variable, the semantics of a derivative variable $\dot{x}$ is the derivative of x (as a function of time) with respect to the time variable. A *differential constraint* over the variables $x_1, \ldots, x_n, \dot{x}_1, \ldots, \dot{x}_n$ has the following general form:

$$\dot{x}_i = f_i(x_1, \ldots, x_n, t), \quad i = 1, \ldots, n$$

where f_i is a multi-variate polynomial with integer coefficients in the variables $x_1, \ldots, x_n$ and t.

Example 2. A differential constraint corresponding to the harmonic oscillator of Example 1 is

$$\dot{x}_1 = f_1(x_1, x_2, t) = x_2$$
$$\dot{x}_2 = f_2(x_1, x_2, t) = -\omega^2 x_1.$$

□

As mentioned in Section 2, a differential constraint does not completely specify an integral curve. Also *initial constraints* on variables x_i and t are needed. We only consider initial constraints of the form

$$I(c_1, \ldots, c_n, t_0) \equiv \bigwedge_{i=1}^{n} x_i = c_i,$$

where the c_is and t_0 are real numbers. Hence, an initial constraint specifies a point in $\mathbb{R}^{n+1}$.

Example 3. Consider the differential constraint given in Example 2. We know already from Example 1 that every integral curve is of the form $(x_1(t), x_2(t))$ where $x_1(t) = A\sin(\omega t + \varphi)$ and $x_2(t) = \dot{x}_1(t)$. Let $c_1, c_2 \in \mathbb{R}$ and consider the initial constraint $I(c_1, c_2, 0)$. The integral curve corresponding to this initial constraint is then uniquely defined by taking $A = \sqrt{(\frac{c_2}{\omega})^2 - c_1^2}$ and $\varphi = \arcsin(\frac{c_1}{\omega})$. □

Definition 2. An n-dimensional *equation of motion* consists of a finite number of triples $(I^{(i)}, DE^{(i)}, \tau^{(i)})$, where for each $i = 1, \ldots, k$,

- $I^{(i)}$ is an initial constraint. I.e., $I^{(i)} = I^{(i)}\left(c_1^{(i)}, \ldots, c_n^{(i)}, t_0^{(i)}\right)$, where $t_0^{(i)} < \tau^{(i)} \leqslant t_0^{(i+1)}$;
- $DE^{(i)}$ is a differential constraint. I.e., $DE^{(i)}$ is equal to $\dot{x}_j = f_j^{(i)}(t, x_1, \ldots, x_n)$, for $j = 1, \ldots, n$; and
- $\tau^{(i)}$ is the final time for which the equation of motion $DE^{(i)}$ is valid.

We denote the set of all n-dimensional equations of motion by $\mathcal{E}$. □

An equation of motion consisting of k initial and differential constraints, corresponds naturally to k integral curves which describe the motion of point during disjoint or adjacent intervals in time.

Example 4. Let DE be the differential constraint given in Example 2. Consider the equation of motion consisting of

$$\{(I(c_1, c_2, 0), DE, 1), (I(d_1, d_2, 1), DE, 2), (I(e_1, e_2, 3), DE, \infty)\}.$$

This equation of motion then corresponds to the description of a moving point using the integral curves in $(x, \dot{x})$-space:

$$t : [0, 2] \cup [3, \infty) \rightarrow \begin{cases} (A_1 \sin(\omega t + \varphi_1), A_1 \omega \cos(\omega t + \varphi_1)) & \text{if } 0 \leq t \leq 1 \\ (A_2 \sin(\omega t + \varphi_2), A_2 \omega \cos(\omega t + \varphi_2)) & \text{if } 1 \leq t \leq 2 \\ (A_3 \sin(\omega t + \varphi_3), A_3 \omega \cos(\omega t + \varphi_3)) & \text{if } 3 \leq t, \end{cases}$$

where the constants A_i and φ_i are determined by the initial constraints as shown in Example 3. We have depicted an instantiation of this example in Figure 1. □

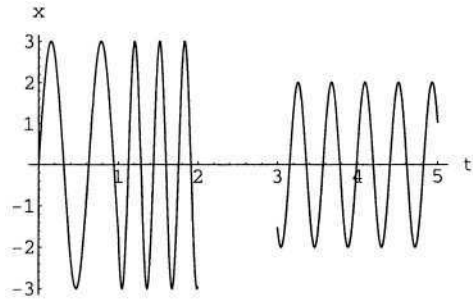

Fig. 1. The integral curves corresponding to the equation of motion given in Example 4 (Only the (x, t)-coordinates are shown).

The definition of equation of motion does not rule out discontinuities in the transition between different integral curves. E.g., in Figure 1 the integral curve disappears between $t = 2$ and $t = 3$. In some applications it might be desirable that integral curves smoothly glue together at transition points. We remark that Definition 2 can easily be restricted to this setting.

We now define a mapping PL from the set of equations of motions $\mathcal{E}$ to the set of trajectories $\mathcal{T}$. With each $e \in \mathcal{E}$ we associate the piecewise linear trajectory $PL(e) \in \mathcal{T}$ obtained by applying a numerical integration method. This mapping is clearly dependent on the choice of numerical integration technique. In this section we will use *Euler's method* [18]. This might not be the best method currently available [10], but suffices for explaining the ideas in this section. We

define the mapping PL in more detail now. Consider first the case that the equation of motion consists of a single initial and differential constraint.

Applied to an initial and a differential constraint I and DE, Euler's method takes a fixed small step size h, sets $\boldsymbol{x}(t_0) = \boldsymbol{c}$, as demanded by the initial constraint and then defines

$$\boldsymbol{x}(t_{i+1}) = \boldsymbol{x}(t_i) + \boldsymbol{f}(\boldsymbol{x}(t_i), t_i)h,$$

where $t_i = t_0 + ih$ for $i = 1, 2, \ldots$ and $\boldsymbol{f} = (f_1, \ldots, f_n)$ are the functions occurring in the differential constraint DE.

One then obtains a trajectory in $\mathcal{T}$ by linearly interpolating between consecutive points. More specifically, if τ is the final time for the differential constraint to be valid, then let K be the maximum integer such that $t_0 + Kh < \tau$ and we define the trajectory $PL((I, DE, \tau))$ as

$$t : (t_0, \tau) \to \begin{cases} \boldsymbol{x}(t_0) + \boldsymbol{f}(\boldsymbol{x}(t_0), t_0)t & \text{if } t_0 \le t \le t_1 \\ \quad\vdots & \quad\vdots \\ \boldsymbol{x}(t_K) + \boldsymbol{f}(\boldsymbol{x}(t_K), t_K)t & \text{if } t_K \le t \le \tau, \end{cases} \tag{3}$$

where again $t_i = t_0 + ih$ for $i = 0, 1, \ldots, K$.

In general, when an equation of motion consists of several triples (I, Eq, τ), the mapping PL is the union of the mappings PL on the individual triples.

We would like to have that if the integral curve is already a trajectory, then the PL mapping will return the same trajectory as well.

Example 5. Consider a point moving along the x-direction between initial time $t_0 = 0$ and final time $t_2 = 1$ such that

$$x(t) = \begin{cases} 2t & \text{if } 0 \le t \le \frac{1}{2} \\ 1 - 2t & \text{if } \frac{1}{2} \le t \le 1. \end{cases} \tag{4}$$

Let $I^{(1)} \equiv x = 0 \wedge t = 0$ and $I^{(2)} \equiv x = 1 \wedge t = \frac{1}{2}$. Furthermore, consider the differential constraints

$$DE^{(1)} \equiv \dot{x} = 2, \qquad DE^{(2)} \equiv \dot{x} = -2.$$

Let e be the equation of motion consisting of the initial constraints $I^{(1)}$ and $I^{(2)}$, together with the corresponding differential constraints $DE^{(1)}$ and $DE^{(2)}$, and final time points $\tau^{(1)} = \frac{1}{2}$ and $\tau^{(2)} = 1$. The trajectory $PL(e)$ is the union of $PL(I^{(i)}, DE^{(i)}, \tau^{(i)})$, $i = 1, 2$ each of which defined by (3). Euler's method on e will now return the same trajectory as represented by (4). □

It is easy to prove that the observation made in the last example holds in general.

Proposition 1. *Let* $\gamma : [t_0, t_1] \to \mathbb{R}^n$ *be a piecewise linear curve. Then there exists an equation of motion* $Eq \in \mathcal{E}_n$ *such that* $PL(Eq) = \gamma$. □

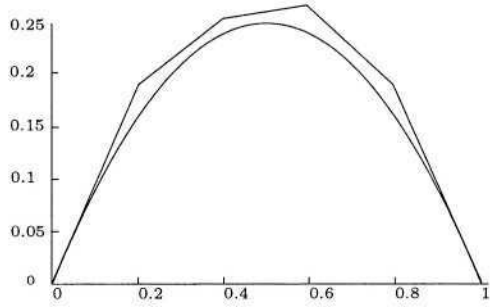

Fig. 2. Integral curve (parabola) of the moving object in Example 6 and corresponding trajectory (linear curve) obtained using a rather large stepsize $h = 0.2$. (Only the (x, y)-coordinates are shown.)

As mentioned above, Euler's method is not the most desirable integration method since it does not provide guarantees on the accuracy. This is common to most step-wise classical numerical integration techniques [20]. In contrast, interval methods produce guaranteed error bounds [16]. They use interval arithmetic to calculate each approximation step, explicitly keeping the error bounds within safe interval bounds. We leave it to future work to integrate interval methods in our framework.

We now have all the ingredients for defining a moving object database. Let $\mathcal{O}$ denote an infinite set of object identifiers.

Definition 3. A n-dimensional *moving object database* is a triple (O, Eq, τ) where O is a finite subset of $\mathcal{O}$, Eq is a mapping from O to $\mathcal{E}$, and τ is a time instance such that each interval in $Eq(o)$ for every object $o \in O$ ends earlier than or at time τ. □

Proposition 1 says that this definition of moving object databases generalizes earlier definitions in which trajectories are explicitly represented by piecewise linear functions. [15, 21]. However, the equation of motion in Proposition 1 requires in general the same number of initial and differential constraints as the number of linear segments of the trajectory and our data model is therefore not necessarily more compact. This changes however when the integral curves are not linear anymore. In these cases a single initial and differential constraint can represent a trajectory consisting of many linear segments.

Example 6. Let e be the equation of motion consisting of an initial constraint $x = 0 \wedge y = 0 \wedge \dot{v} = 1 \wedge \dot{w} = 1 \wedge t = 0$, together with the differential constraint

$$\dot{x} = v, \quad \dot{y} = w, \quad \dot{v} = 0, \quad \dot{w} = -g,$$

and a final point in time $\tau = 1$. A simple computation shows that the solution of this equation of motion is

$$t \mapsto (t, -gt^2 + t, 1, -2gt + 1),$$

which restricted to the (x, y)-plane results in a parabola. By definition *PL* will map this equation of motion to a linear approximation of the parabola. We have depicted this situation in Figure 2. We used a large stepsize in Euler's method in order to distinguish the integral curve from its approximating trajectory. □

We remark that our definition of differential constraints often demands the introduction of new variables. It might be therefore desirable to make explicit in which dimension of the integral curve one actually is interested in (e.g., three spatial dimensions and time). This can be easily added to the definition of equations of motion.

4 Physical Correctness

Updates for moving objects are of particular importance due to their dynamic character. We therefore need to be able to update moving object databases and the following pair of updates adapted from [15] clearly suffices.

Definition 4. Let τ be a time point, $o \in \mathcal{O}$, $e \in \mathcal{E}_n$. An *update* on a moving object database (O, Eq, τ) is one of the following:

- Create a new object: $\mathsf{new}(o, \tau', e)$ results in $(O \cup \{o\}, Eq', \max\{\tau, \tau'\})$ where Eq is identical to Eq except for $Eq(o) = e$.
- Terminate an existing object: $\mathsf{terminate}(o, \tau')$ results in (O, Eq', τ) where Eq' is identical to Eq except for $Eq'(o) = Eq(o) \wedge t \leq \tau'$. □

In addition to the usual demands for updates, the physical meaning of the equations of motions also demands the presence of update operators on the moving object databases. Indeed, consider two moving objects o_1 and o_2 whose integral curves intersect at time t_c. We say that o_1 and o_2 are in contact at time t_c. At this contact time t_c the initial and differential constraints should change according to physical laws in order to represent a physical reality. We call a moving object database *physically consistent* if at times of contact the equations of motions have been updated correctly. We will not give explicitly the formulas needed for the updates but refer instead to the tutorial of Baraff for derivation and explanation of the updates [4].

One distinguishes between two types of contact. When o_1 and o_2 are in contact at time t_c and they have a velocity towards each other, we speak of a *colliding contact.* Colliding contacts require an instantaneous change in velocity of the objects. This implies that at time t_c the differential constraint still holds but the initial constraint needs to be updated.

When o_1 and o_2 are in contact at time t_c and o_2 is resting or cannot move, e.g. in case o_2 is a wall, then we speak of a *resting contact.* In this case, one needs to compute a contact force and has to change the differential constraint accordingly in the equation of motion of o_1 for $t \geq t_c$.

Note that we assumed that the contact times t_c are known. Collision detection algorithms provide a way of computing (or approximating) these intersection points. It has been extensively studied in computational geometry and

robotics [13] and currently fast and robust algorithms exists. However, most of these algorithms are practical for a small number of moving objects. Also it is difficult when moving objects are geometric objects other than points and are deformable. We refer to the survey on modeling of motion for the current state of research [1].

5 Query Language

Now that we have defined the data model, it is time to look for a query language. We first give the kind of queries which we would like to ask and answer. A very natural example of a moving database is an interplanetary database containing moving planets, galaxies, space shuttles, satellites and so forth. We are now interested in the following kind of queries.

Q_1 Locate all planets in the earth's North hemisphere at 2pm.
Q_2 Find all comets entering our solar system today.
Q_3 List all pairs of satellites flying in opposite direction.
Q_4 List all moons of planets in our solar system which are visible at least 10 minutes today.
Q_5 Give all pairs of planets and asteroids which are in collision course.

In order to ask and answer these queries we will use the query languages FO(f) based on a generalized distance f as defined by Mokthar et al. [15].

Let Γ denote the set of all continuous functions from $\mathbb{R}$ to $\mathbb{R}^n$.

Definition 5. A *generalized distance* is a mapping from the set Γ of continuous curves to continuous functions from $\mathbb{R}$ to $\mathbb{R}$. Let a (O, Eq, τ) be a moving database and f a generalized distance. For each $o \in O$ we set $\mathsf{f}_o = \mathsf{f}(\gamma)$, where γ is the integral curve corresponding to $Eq(o)$. Moreover, we define $\hat{\mathsf{f}}_o = \mathsf{f}(PL(Eq(o)))$. □

So, a generalized distance describes some property of curves which continuously changes in time.

The language FO(f) consists of a many-sorted first-order logic with real numbers, time instants, and objects. The language uses a single time variable t and many objects variables. We do not allow real variables; these are embedded in the generalized functions.

- *time terms* are polynomials over the time variable t with integer coefficients.
- *real terms* include real variables, and $\mathsf{f}(y, t)$ where y is an object variable and t is a time term.

Atomic formulas are formed by equality and order predicates over terms of the same sort. Formulas are then constructed by propositional connectives and universal/existential quantifiers over object variables.

Definition 6. A query is a quadruple (y, t, I, φ) where y is an object variable, t a time variable, I a time interval and φ a formula with only y and t free. □

Let $D = (O, Eq, \tau_0)$ be a moving object database. Then for each time τ, we define

$$Q[D]_\tau = \{(o) \mid o \in O \wedge \varphi(o, \tau)\}.$$

The answer to Q can then be of an *existential* nature, $Q^\exists(D) = \{(o) \mid \exists t(t \in I \wedge o \in Q[D]_t)\}$, or *universal,* $Q^\forall(D) = \{(o) \mid \forall t(t \in I \to o \in Q[D]_t)\}$. It is clear that both $Q^\exists$ and $Q^\forall$ can be obtained from Q_τ. It is easy to see that queries $Q_1, \ldots, Q_5$ can all be expressed in FO(f).

6 Evaluating the Queries

The evaluation procedure of FO(f) queries in our database model is an adaptation of the procedure given by Mokthar et al. [15]. Let $Q = (y, t, I\varphi)$ be a query and D a moving object database. Mokhtar et al. showed that in order to evaluate $Q[D]_\tau$ it is sufficient to know at each time $t \in I$ the precedence relation $\preceq_t$ defined by

Definition 7. Let $D = (O, Eq, \tau')$ be a moving object database, τ a time instant, $o, o' \in O$. The object o *precedes* o' at time τ, denoted by $o \preceq_\tau o'$, if $\mathsf{f}_o(\tau) \leq \mathsf{f}_{o'}(\tau)$. □

If the moving objects are represented by trajectories, then the precedence relation can be easily obtained as shown in [15].

In our data model, we have a larger class of trajectories consisting of the integral curves of the equations of motion. Since we do not have the integral curves at our disposal we will replace them for the time period I by the trajectories obtained by applying the PL map defined in Section 3.

Definition 8. Let $D = (O, Eq, \tau')$ be a moving object database, τ a time instant, $o, o' \in O$. The object o *approximately precedes* o' at time τ, denoted by $o \preceq_\tau o'$, if $\hat{\mathsf{f}}_o(\tau) \leq \hat{\mathsf{f}}_{o'}(\tau)$. □

Instead of answering Q based on the precedence relation $\preceq_\tau$ we will answer it based on the approximate precedence relation $\preceq_\tau$. In order to guarantee a good approximation we only have to be sure that the intersection points of the curves

$$\gamma : t \in I \to \mathsf{f}_o(t), \quad o \in O$$

are accurately approximated by the intersection points of

$$\gamma : t \in I \to \hat{\mathsf{f}}_o(t), \quad o \in O,$$

since these determine possible changes in the precedence relations.

If we simply take the *PL* map defined in Section 3 based on Euler's method then this will not be the case since this method has fixed time step size h and will therefore make wrong conclusion concerning the intersection of the above curves. Using standard approaches from physically based modeling, we can however change step size in the neighborhood of intersection points of the above curves.

Two common approaches exists. In retroactive detection [3] one checks after each integration step for intersection points. If an intersection is detected, one traces back the exact moment of intersection using e.g., bisection methods or regular falsa, and the integration starts again from the intersection moment. On the other hand, conservative advancement methods [5] creep up to intersection methods, taking smaller steps as it gets closer. For our purposes it does not really matter which method is used. It only matters when there are a vast number of moving objects in the database. In this case one can rely on the Timewarp algorithm [14].

The above methods are assumed to be part of the PL mapping so that the approximate precedence relation captures the exact precedence relations as good as possible. Using this approximate precedence relation, a query in FO(f) can be evaluated as described by Mokthar et al. [15].

7 Conclusions

We proposed a new data model for moving object based on equations of motions and showed that the query language proposed by Mokthar et al. [15] can be used also in our setting. The evaluation of queries is however different since trajectories are not stored in the database but are generated in real-time when a query is asked. These trajectories can be generated with an adjustable degree of accuracy depending on the database instance. Using techniques from physically based modeling we're able to guarantee a successful answer to these queries. Many aspects still need to be investigated, including

- The complexity of the computation of the approximate precedence relation, and more general the complexity of the query evaluation described in Section 6.
- The generalization of the moving point objects to general moving objects using results from [4]. This involves a more complex definition of differential constraints and update operators. A challenge is to extend the generalized distance-base query language FO(f) so that it can query the spatial extent of the moving object as well.
- Data integration of our data model. It is important that the proposed data model integrates well with other spatio-temporal models.
- The automatic derivation of simple equations of motion from observed trajectories.

Acknowledgement

The author would like to thank Taneli Mielikäinen for his encouraging remarks.

References

1. P. K. Agarwal, L.J. Guibas, H. Edelsbrunner, J. Erickson, M. Isard, S. Har-Peled, J. Hershberger, C. Jensen, L. Kavraki, P. Koehl, M. Lin, D. Manocha, D. Metaxas, B. Mirtich, D. Mount, S. Muthukrishnan, D. Pai, E. Sacks, J. Snoeyink, S. Suri, and O. Wolefson. Algorithmic issues in modeling motion. *ACM Comput. Surv.,* 34(4):550–572, 2002.
2. V. I. Arnold, A. Weinstein, and K. Vogtmann. *Mathematical Methods of Classical Mechanics,* volume 60 of *Graduate Texts in Mathematics.* Springer-Verlag, 1997.
3. D. Baraff. Curved surfaces and coherence for non-penetrating rigid body simulation. *Computer Graphics,* 24(4):19–28, 1990.
4. D. Baraff and A. Witkin. *Physically Based Modelling.* Pixar Animation studios, 2001. SIGGRAPH 2001 Course Notes.
5. J. Basch, L.J. Guibas, and J. Hershberger. Data structures for mobile data. *Journal of Algorithms,* 31(3):1–28, 1999.
6. J. Chomicki and P. Revesz. Constraint-based interoperability of spatiotemporal databases. *Geoinformatica,* 3(3):211–2423, 1999.
7. M. Erwig, R.H. Güting, M. Schneider, and M. Vazirgiannis. Spatio-temporal data types: An approach to modeling and querying moving objects in databases. *Geoinformatica,* 3(3):269–296, 1999.
8. R.P. Feynman, R.B. Leighton, and M Sands. *The Feynman Lectures on Physics.* Addison-Wesley, 1989.
9. R. H. Güting, M.H. Böhlen, M. Erwig, C.S. Jensen, N.A. Lorentzos, M. Schneider, and M. Vazirgiannis. A foundation for representing and querying moving objects. *ACM Trans. Database Syst.,* 25(1):1–42, 2000.
10. A. Iserles, editor. *A First Course in the Numerical Analysis of Differential Equations.* Cambridge University Press, 1996.
11. P. C. Kanellakis, G. M. Kuper, and P.Z. Revesz. Constraint query languages. *Journal of Computing and Systems Sciences,* 51(1):26–52, 1995.
12. G.M. Kuper, J. Paredaens, and L. Libkin, editors. *Constraint Databases.* Springer-Verlag, 2000.
13. M. Lin and S. Gottschalk. Collision detection between geometric models. In *Proceedings of the IMA Conference on Mathematics of Surfaces,* 1998.
14. B. Mirtich. Timewarp rigid body simulation. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques,* pages 193–200, 2000.
15. H. Mokthar, J. Su, and O. Ibarra. On moving object queries. In *Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems,* pages 188–198. ACM Press, 2002.
16. N.S. Nedialkov. *Computing Rigorous Bounds on the Solution of an Initial Value Problem for an Ordinary Differential Equation.* PhD thesis, Computer Science Dept., University of Toronto, 1999.
17. I. Newton. *Philosophiae Naturalis Principia Mathematica.* 1687.
18. W.H. Press, Flannery B.P., Teukolsky S.A., and Vetterling W.T. *Numerical Recipes in C : The Art of Scientific Computing.* Cambridge University Press, 1992.
19. P. Revesz. *Introduction to Constraint Databases.* Springer-Verlag, 2001.
20. L.F. Shampine. *Numerical Solution of Ordinary Differential Equations.* Chapman & Hall, 1994.
21. J. Su, H. Xu, and O.H. Ibarra. Moving objects: Logical relationships and queries. In *Proceedings of the 7th International Symposium on Advances in Spatial and Temporal Databases,* volume 2121 of *Lecture Notes in Computer Science,* pages 3–19. Springer, 2001.

A Triangle-Based Logic for Affine-Invariant Querying of Two-Dimensional Spatial Data

Sofie Haesevoets

Limburgs Universitair Centrum
Department WNI
Universitaire Campus, B-3590 Diepenbeek, Belgium
sofie.haesevoets@luc.ac.be

Abstract. In spatial databases, incompatibilities often arise due to different choices of origin or unit of measurement (e.g., centimeters versus inches). By representing and querying the data in an affine-invariant manner, we can avoid these incompatibilities.
In practice, spatial (resp., spatio-temporal) data is often represented as a finite union of triangles (resp., moving triangles). As two arbitrary triangles are equal up to a unique affinity of the plane, they seem perfect candidates as basic units for an affine-invariant query language.
We propose a so-called "triangle logic", a query language that is affine-generic and has triangles as basic elements. We show that this language has the same expressive power as the affine-generic fragment of first-order logic over the reals on triangle databases. We illustrate that the proposed language is simple and intuitive. It can also serve as a first step towards a "moving-triangle logic" for spatio-temporal data.

1 Introduction and Summary of the Results

In the area of spatial database research, a lot of attention has been focused on affine invariance of both data and queries. The main purpose of studying affine invariance is to obtain methods and techniques that are not affected by affine transformations of the input spatial data. This means that a particular choice of origin or some artificial choice of unit of measure (e.g., inches versus centimeters) has no effect on the final result of the method or query.

In computer vision, the so-called *weak perspective assumption* [22] is widely adopted. When an object is repeatedly photographed under different camera angles, all the different images are assumed to be affine transformations of each other. This assumption led to the need for affine-invariant similarity measures between pairs of pictures [10,11,15]. In computer graphics, affine-invariant norms and triangulations have been studied [17]. In the field of spatial and spatio-temporal constraint databases [20,21], affine-invariant query languages [7,8] have been proposed. In Section 2, we will go into more detail about the affine-invariant language for spatial constraint data proposed by Gyssens, Van den Bussche and Van Gucht [8]. Affinities are one of the transformation groups proposed at the introduction of the concept of "genericity of query languages" applied to constraint databases [19]. Also various subgroups of the affinities [19]

B. Kuijpers and P. Revesz (Eds.): CDB 2004, LNCS 3074, pp. 52–73, 2004.

such as isometries, similarities, … and supergroups of the affinities such as topology preserving transformations [13, 18] have been studied in the same context.

If we now focus on the representation of two-dimensional spatial data, we see that, in practice, two-dimensional figures are approximated often as a finite union of triangles. In geographic information systems, Triangulated Irregular Networks (TIN) [16] are often used. In computer graphics, data is approximated by triangular meshes (e.g., [2]). Also, for spatio-temporal databases, "parametric moving triangle"-based models have been proposed and studied [4, 5].

Remark that two arbitrary triangles are indistinguishable up to an affinity of the plane. Indeed, each triangle in the plane can be mapped to each other triangle in the plane by a unique affinity[1].

The combination of the need for affine-invariance, the representation of data by means of triangles in practice, and the fact that triangles itself are an affine-invariant concept, led to the idea of introducing a query language based on triangles. If the data is represented as a collection of triangles, why should one reason about it as a collection of points [8], or, even indirectly, by means of coordinates (as is the case for the classical spatial constraint language, first-order logic over the reals)? We consider first-order languages, in which variables are interpreted to range over triangles in the plane, as they are shown to have many nice properties in the setting of constraint databases [20].

We propose a new, first-order query language that has triangles as basic elements. We show that this language has the same expressive power as the affine-invariant segment of the queries in first-order logic over the reals on triangle databases. Afterwards, we give some examples illustrating the expressiveness of our language. We also address the notion of safety of triangle queries. We show that it is undecidable whether a specific triangle query returns a finite output on finite input. It is, however, decidable whether the output of a query on a particular finite input database can be represented as a finite union of triangles. We show that we can express this finite representation in our triangle language.

The outline of this article is as follows. In Section 2, we start with related work and give some preliminary definitions. Then we define triangle databases, relations and queries in Section 3. In Section 4, we show that our triangle-based logic has the same expressiveness as the affine-generic queries expressible in first-order logic over the reals on triangle databases. We illustrate the expressiveness of our language on a real-world example in Section 5. In Section 6, we address the important question whether we can decide whether the output of a triangle query on a finite input returns finite output. We conclude with a summary and directions for further work.

2 Related Work and Preliminaries

The idea that the result of a query on some spatial input database should be invariant under some group of spatial transformations, was first introduced by

[1] This is only true if the triangle is not degenerated, i.e., no corner points coincide. Otherwise, there are more such affinities.

Paredaens, Van den Bussche and Van Gucht [19]. In a follow-up article, Gyssens, Van den Bussche and Van Gucht [8] proposed several first-order query languages, invariant under group of the affinities or some subgroup thereof. In these languages, variables are assumed to range over points in some real space $\mathbf{R}^n$ ($\mathbf{R}$ is the set of real numbers), rather than over real numbers (coordinates of such points). For the group of the affinities, the point language with only one predicate that expresses *betweenness* of points, was shown to have the same expressivity as the affine-invariant fragment of first-order logic over the reals, on point databases. We will use this result to prove the expressiveness of our triangle-based logic. Therefore, we will recall some definitions from the article from Gyssens, Van den Bussche and Van Gucht [8]. All definitions listed in this section can be found there.

We start with the well-known definition of a constraint database, or semi-algebraic database, as this is the general setting which we will be working in.

Definition 1. A *semi-algebraic relation in* $\mathbf{R}^n$ is a subset of $\mathbf{R}^n$ that can be described as a Boolean combination of sets of the form

$$\{(x_1, x_2, \ldots, x_n) \in \mathbf{R}^n \mid p(x_1, x_2, \ldots, x_n) > 0\},$$

with p a polynomial with integer coefficients in the real variables $x_1, x_2, \ldots, x_n$. □

In mathematical terms, semi-algebraic relations are known as *semi-algebraic sets* [3].

We also call a semi-algebraic relation in $\mathbf{R}^n$ *a semi-algebraic relation of arity* n. A semi-algebraic database is essentially a finite collection of semi-algebraic relations. We give the definition next.

Definition 2. A *(semi-algebraic) database schema* σ is a finite set of relation names, where each relation name R has an arity associated to it, which is a natural number and which is denoted by $ar(R)$.

Let σ be a database schema. A *semi-algebraic database over* σ is a structure $\mathcal{D}$ over σ with domain $\mathbf{R}$ such that, for each relation name R of σ, the associated relation $R^{\mathcal{D}}$ in $\mathcal{D}$ is a semi-algebraic relation of arity $ar(R)$. □

Example 1. Let $\sigma = \{R, S\}$, with $ar(R) = 2$ and $ar(S) = 1$ be a semi-algebraic database schema. Then the structure $\mathcal{D}$ given by

$$(\mathbf{R}, R^{\mathcal{D}} = \{(x_1, x_2) \in \mathbf{R}^2 \mid x_1^2 + x_2^2 < 1\}, S^{\mathcal{D}} = \{x \in \mathbf{R} \mid 0 \leq x \leq 1\})$$

is an example of a semi-algebraic database over σ that contains the open unit disk and the closed unit interval. □

Definition 3. Let σ be a n-dimensional semi-algebraic database schema. The language FO[+,×,<,0,1, σ] (or FO[+,×,<,0,1], if σ is clear from the context), first-order logic over the real numbers with polynomial constraints, is the first-order language with variables that are assumed to range over real numbers,

where the atomic formulas are either of the form $p(x_1, x_2, \ldots, x_n) > 0$, with p a polynomial with integer coefficients in the real variables $x_1, x_2, \ldots, x_n$, or the relation names from σ applied to real terms. Atomic formulas are composed using the operations $\wedge$, $\vee$ and $\neg$ and the quantifiers $\forall$ and $\exists$. □

Example 2. Consider the semi-algebraic database from Example 1. The expression

$$R(x, y) \wedge y > 0$$

is a FO[+,x,<,0,1]-formula selecting the part of the open unit disk that lies strictly above the x-axis. □

We restrict all further definitions and results to dimension $n = 2$, as this is the dimension we will be working with in the rest of this text, although they were originally proved to hold for arbitrary n, $n \geq 2$.

Now we give the definition of a geometric database, a special type of constraint database that contains a possibly infinite number of points.

Definition 4. Let σ be a geometric database schema. A *geometric database* over σ in $\mathbf{R}^2$ is a structure $\mathcal{D}$ over σ with domain $\mathbf{R}^2$ such that, for each relation name R of σ, the associated relation $R^{\mathcal{D}}$ in $\mathcal{D}$ is semi-algebraic. □

A geometric database $\mathcal{D}$ over σ in $\mathbf{R}^2$ can be viewed naturally as a semi-algebraic database $\overline{\mathcal{D}}$ over the schema $\overline{\sigma}$, which has, for each relation name R of σ, a relation name $\overline{R}$ with arity $2k$, where k is the arity of R in σ. For each relation name R, of arity k, $\overline{R}^{\overline{\mathcal{D}}}$ is obtained from $R^{\mathcal{D}}$ by applying the *canonical bijection*[2] between $(\mathbf{R}^2)^k$ and $\mathbf{R}^{2k}$.

Definition 5. Let σ be a geometric database schema. A *k-ary geometric query* Q over σ in $\mathbf{R}^2$ is a partial computable function on the set of geometric databases over σ. Furthermore, for each geometric database $\mathcal{D}$ over σ on which Q is defined, $Q(\mathcal{D})$ is a geometric relation of arity k. □

Queries that are invariant under some transformation group G of $\mathbf{R}^2$, are also called *G-generic* [19]. We define this next:

Definition 6. Let σ be a geometric database schema and Q a geometric query over σ in $\mathbf{R}^2$. Let G be a group of transformations of $\mathbf{R}^2$. Then Q is called *G-generic* if, for any two geometric databases $\mathcal{D}$ and $\mathcal{D}'$ over σ in $\mathbf{R}^2$ for which $\mathcal{D}' = g(\mathcal{D})$, for some $g \in G$, we have that $Q(\mathcal{D}') = g(Q(\mathcal{D}))$. □

In the remainder of this text, we will only focus on the group G of *affinities.* The affinities of $\mathbf{R}^2$ form the group of linear transformations having a regular

[2] The canonical bijection between $(\mathbf{R}^2)^k$ and $\mathbf{R}^{2k}$ associates with each k-tuple $(\mathbf{x_1}, \ldots, \mathbf{x_k})$ of $(\mathbf{R}^2)^k$ the $2k$-tuple $(x_1^1, x_1^2, \ldots, x_k^1, x_k^2)$, where for $1 \leq i \leq k$ $\mathbf{x_i} = (x_i^1, x_i^2)$.

matrix, i.e., their matrix has a determinant different from zero. Affinities of the plane have the following form:

$$\begin{pmatrix} x \\ y \end{pmatrix} \mapsto \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} e \\ f \end{pmatrix},$$

where $ad - bc$ is different from zero.

We now give the definition of the first-order point logic FO[**Between**], a first-order language where the variables are not interpreted as real numbers, as in FO[+,×,<,0,1], but as 2-dimensional points.

We first introduce the point predicate **Between**.

Definition 7. Let $p = (p_x, p_y)$, $q = (q_x, q_y)$ and $r = (r_x, r_y)$ be points in the plane. The expression $\mathbf{Between}(p, q, r)$ is true iff either q lies on the closed line segment between p and r or p and/or q and/or r coincide. □

In Figure 1, $\mathbf{Between}(p, t, q)$, $\mathbf{Between}(p, p, q)$ and $\mathbf{Between}(t, s, r)$ are true, but $\mathbf{Between}(t, q, p)$ and $\mathbf{Between}(p, q, r)$ are not.

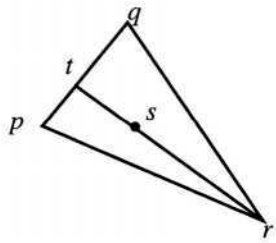

Fig. 1. The predicate **InTriangle** can be expressed using **Between**.

Definition 8. Let σ be a 2-dimensional geometric database schema. The first-order point language over σ and **Between**, denoted by FO[**Between**, σ] (or FO[**Between**], if σ is clear from the context), is a first-order language with variables that range over points in $\mathbf{R}^2$, (denoted $\hat{p}, \hat{q}, \ldots$), where the atomic formulas are equality constraints on point variables, the predicate **Between** applied to point variables, and the relation names from σ applied to point variables. □

A FO[**Between**]-formula $\varphi(\hat{p}_1, \hat{p}_2, \ldots, \hat{p}_l)$ over the relation names of σ and the predicate **Between** defines on each geometric database $\mathcal{D}$ over σ a subset $\varphi(\mathcal{D})$ of $(\mathbf{R}^2)^l$ in the standard manner.

Gyssens, Van den Bussche and Van Gucht showed that the query language FO[**Between**] expresses exactly all affine-generic geometric queries expressible in FO[+,×,<,0,1].

3 A Triangle-Based Logic

In practice, spatial data models often represent data as a collection of triangles (TINs for example [16]). Therefore, it seems natural to have a query language that manipulates triangles instead of points or coordinates. Triangles can be represented as triples of points. We consider triangles to be filled. Corner points are allowed to coincide, so lines and points are also considered triangles. If we list the corner points of a triangle, the order in which we list them is arbitrary.

3.1 Definitions

In this section we define triangle databases and queries on triangle databases.

We start with the definition of a *Tbase,* a database that contains a possibly infinite collection of triangles.

Definition 9. Let σ be a database schema. A *Tbase* over σ in $(\mathbf{R}^2)^3$ is a structure $\mathcal{D}$ over σ with domain $(\mathbf{R}^2)^3$ such that, for each relation name R of σ, the associated triangle relation $R^{\mathcal{D}}$ in $\mathcal{D}$ is a geometric relation. □

We will refer to a relation in a Tbase as a *Trelation.*

A Tbase $\mathcal{D}$ over σ in $(\mathbf{R}^2)^3$ can be viewed naturally as a geometric database $\hat{\mathcal{D}}$ over the schema $\hat{\sigma}$, which has, for each relation name R of σ, a relation name $\hat{R}$ with arity $3k$, where k is the arity of R in σ. For each relation name R, of arity k, $\hat{R}^{\hat{\mathcal{D}}}$ is obtained from $R^{\mathcal{D}}$ by applying the canonical bijection[3] between $((\mathbf{R}^2)^3)^k$ and $(\mathbf{R}^2)^{3k}$.

Definition 10. Let σ be a database schema. A *k-ary Tquery* Q over σ (in $(\mathbf{R}^2)^3$) is a partial computable function on the set of Tbases over σ such that, for each Tbase $\mathcal{D}$ over σ on which Q is defined, $Q(\mathcal{D})$ is a Trelation of arity k. □

We now introduce a triangle-based logic.

Definition 11. Let σ be a triangle database schema and let $\mathbf{\Delta}$ be a set of predicates of a certain arity over triangles in $(\mathbf{R}^2)^3$. The first-order triangle language over σ and $\mathbf{\Delta}$, denoted by $\mathrm{FO}[\mathbf{\Delta}, \sigma]$ (or $\mathrm{FO}[\mathbf{\Delta}]$ if σ is understood from the context), is a first-order language with variables that range over triangles in $(\mathbf{R}^2)^3$, (denoted $\Delta_p, \Delta_q, \ldots$), where the atomic formulas are equality constraints on triangle variables, the predicates of $\mathbf{\Delta}$ applied to triangle variables, and the relation names from σ applied to triangle variables. □

A first-order formula $\varphi(\Delta_{p_1}, \Delta_{p_2}, \ldots, \Delta_{p_l})$ over the relation names of σ and the predicate names in $\mathbf{\Delta}$ defines on each Tbase $\mathcal{D}$ over σ a subset $\varphi(\mathcal{D})$ of $((\mathbf{R}^2)^3)^l$ in the standard manner.

[3] The canonical bijection between $((\mathbf{R}^2)^3)^k$ and $(\mathbf{R}^2)^{3k}$ associates with each k-tuple $(\mathbf{x_1}, \ldots, \mathbf{x_k})$ of $(\mathbf{R}^6)^k$ the $3k$-tuple $(x_1^1, x_1^2, x_1^3, \ldots, x_k^1, x_k^2, x_k^3)$, where for $1 \leq i \leq k$ $\mathbf{x_i} = (x_i^1, x_i^2, x_i^3)$.

We will sometimes use the symbol $=_\Delta$ to indicate equality of triangle variables, as opposed to equality of point variables. If it is clear from the context which type of variables is used in the formula, we will omit the index.

We define the following triangle predicates (see also Figure 2):

Definition 12.
- $\mathbf{PartOf}(\Delta_p, \Delta_q)$ denotes that the set of points in $\mathbf{R}^2$ that compose the triangle Δ_p is a subset of the points that compose triangle Δ_q;
- $\mathbf{Line}(\Delta_p)$ denotes that the triangle Δ_p is degenerated into a line segment, i.e., exactly two corner points coincide;

□

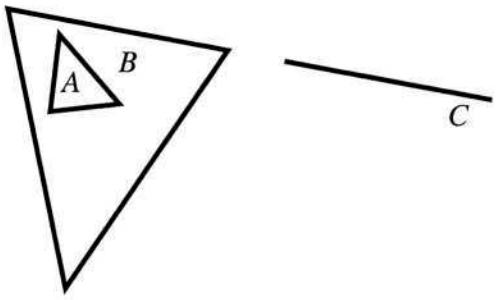

Fig. 2. In this figure, the predicates **PartOf**(*A, B*) and **Line**(*C*) are true.

From now on, in the remainder of this article, we take

$$\mathbf{\Delta} = \{\mathbf{PartOf}, \mathbf{Line}\}.$$

3.2 Examples of Derived Triangle Predicates

We also derive some other predicates, which will be used in later sections. We abbreviate expressions of the form $(\exists a)(\exists b)(\exists c)\ldots$ by $(\exists a, b, c, \ldots)$.

- The expression $\Delta_p \neq \Delta_q$ is an abbreviation for $\neg(\Delta_p = \Delta_q)$.
- The predicate **SameCarrier** expresses that two (triangles degenerated into) line segments lie on the same line. The expression $\mathbf{SameCarrier}(\Delta_p, \Delta_q)$ stands for

$$\mathbf{Line}(\Delta_p) \wedge \mathbf{Line}(\Delta_q) \wedge (\exists \Delta_r)(\mathbf{Line}(\Delta_r) \wedge \mathbf{PartOf}(\Delta_p, \Delta_r) \wedge \mathbf{PartOf}(\Delta_q, \Delta_r)).$$

- The predicate **Point** indicates that a triangle is degenerated into a point. The definition of $\mathbf{Point}(\Delta_p)$ is

$$(\exists \Delta_q, \Delta_r)(\mathbf{Line}(\Delta_q) \wedge \mathbf{Line}(\Delta_r) \wedge \neg(\Delta_q = \Delta_r) \wedge \neg\mathbf{SameCarrier}(\Delta_q, \Delta_r) \wedge \mathbf{PartOf}(\Delta_p, \Delta_q) \wedge \mathbf{PartOf}(\Delta_p, \Delta_r)).$$

- The predicate **LineAdj** expresses that two triangles have exactly one line segment in common. The expression $\mathbf{LineAdj}(\Delta_p, \Delta_q)$ stands for

$$\neg\mathbf{Point}(\Delta_p) \wedge \neg\mathbf{Point}(\Delta_q) \wedge (\exists\Delta_r)(\mathbf{Line}(\Delta_r) \wedge (\forall\Delta_s)((\mathbf{PartOf}(\Delta_s, \Delta_p) \wedge \mathbf{PartOf}(\Delta_s, \Delta_q)) \Leftrightarrow \mathbf{PartOf}(\Delta_s, \Delta_r))).$$

- The predicate **RealTriangle** expresses that no corner points of a triangle coincide. The expression $\mathbf{RealTriangle}(\Delta_p)$ stands for

$$\neg(\mathbf{Point}(\Delta_p) \vee \mathbf{Line}(\Delta_p)).$$

- The predicate **CornerP** expresses that three points (triangles degenerated into a point) are the corner points of a (real) triangle. The formula

$$\mathbf{RealTriangle}(\Delta_p) \wedge \bigwedge_{i=1}^{3} \mathbf{Point}(\Delta_{p_i}) \wedge (\exists\Delta_{c_1}, \Delta_{c_2}, \Delta_{c_3})$$
$$(\Delta_{c_1} \neq \Delta_{c_2} \wedge \Delta_{c_1} \neq \Delta_{c_3} \wedge \Delta_{c_2} \neq \Delta_{c_3} \wedge \bigwedge_{i=1}^{3} \mathbf{Line}(\Delta_{c_i})$$
$$\wedge \bigwedge_{i=1}^{3} \mathbf{LineAdj}(\Delta_{c_i}, \Delta_p) \wedge \bigwedge_{i=1}^{3} (\mathbf{PartOf}(\Delta_{p_i}, \Delta_{c_i}) \wedge$$
$$\mathbf{PartOf}(\Delta_{p_i}, \Delta_{c_{(i+1)\mathrm{Mod}3}})))$$

  expresses $\mathbf{CornerP}(\Delta_{p_1}, \Delta_{p_2}, \Delta_{p_3}, \Delta_p)$.
- The predicate **Intersect** expresses that two triangles intersect. The definition of $\mathbf{Intersect}(\Delta_p, \Delta_q)$ is

$$(\exists\Delta_r)(\mathbf{PartOf}(\Delta_r, \Delta_p) \wedge \mathbf{PartOf}(\Delta_r, \Delta_q)).$$

- The predicate **EndP** expresses that two points (triangles degenerated into a point) are the end points of a (triangle degenerated into a) line segment. The expression $\mathbf{EndP}(\Delta_p, \Delta_q, \Delta_r)$ stands for the formula

$$\mathbf{Point}(\Delta_p) \wedge \mathbf{Point}(\Delta_q) \wedge \mathbf{Line}(\Delta_r) \wedge \mathbf{PartOf}(\Delta_p, \Delta_r) \wedge$$
$$\mathbf{PartOf}(\Delta_q, \Delta_r) \wedge \Delta_p \neq \Delta_q \wedge (\exists\Delta_s, \Delta_t)(\mathbf{LineAdj}(\Delta_s, \Delta_r) \wedge$$
$$\mathbf{RealTriangle}(\Delta_s) \wedge \mathbf{CornerP}(\Delta_p, \Delta_q, \Delta_t, \Delta_s)).$$

4 Expressiveness of FO[Δ]

We now determine the expressiveness of the language FO[Δ]. We prove that it has the same expressiveness as the affine-invariant fragment of fist-order logic over the reals, on Tbases. We prove this by comparing FO[Δ] and FO[**Between**]. We prove that FO[Δ] has the same expressiveness as FO[**Between**] on Tbases.

Remark that, as we defined a Tbase as a special type of geometric database, we take the affine image of a triangle for affinities of the plane, and not of $\mathbf{R}^6$.

This corresponds to our intuition of the affine image of a triangle. Indeed, the affine image of a triangle *pqr* under some affinity α of the plane, is the triangle $\alpha(p)\alpha(q)\alpha(r)$.

The proof that $\mathrm{FO}[\Delta]$ is as expressive as FO[**Between**] on Tbases has two parts: one that proves *soundness*, and one that proves *completeness* of $\mathrm{FO}[\Delta]$ relative to FO[**Between**]. A language is sound for the FO[**Between**]-Tqueries, if that language only expresses FO[**Between**]-Tqueries on Tbases. A language is complete for the FO[**Between**]-Tqueries, if all FO[**Between**]-Tqueries can be expressed in that language.

The proofs of the next two lemma's can be found in the Appendix.

Lemma 1. The language $\mathrm{FO}[\Delta]$ is *sound* for the affine-generic first-order geometric queries over Tbases. □

Lemma 2. The language $\mathrm{FO}[\Delta]$ is *complete* for the affine-generic geometric queries over Tbases. □

We now combine the soundness and completeness lemmas, and use them to prove our main theorem:

Theorem 1. The language $\mathrm{FO}[\Delta]$ is sound and complete for the affine-generic FO[+,×,<,0,1]-queries on Tbases. □

Proof. From Lemma 1 and Lemma 2, we can conclude that $\mathrm{FO}[\Delta]$ is sound and complete for the FO[**Between**]-queries on Tbases.

Gyssens, Van den Bussche and Van Gucht showed that FO[**Between**] is sound and complete for the affine-generic FO[+,×,<,0,1]-queries on geometric databases [8].

From the definition of Tbases, we know that they are a subset of the geometric databases. This concludes the proof. □

The following property follows from the fact that FO[**Between**]**+While** is sound and complete for the computable affine-generic queries on geometric databases [8]. The language FO[**Between**]**+While** is a language in which FO[**Between**]-definable relations can be created and which has a while-loop with FO[**Between**]-definable stop conditions.

Corollary 1. The language $\mathrm{FO}[\Delta]+\mathbf{While}$ is sound and complete for the computable affine-generic queries on Tbases. □

5 Examples

5.1 A Few More Triangle Predicates

Except from the predicates listed in Section 3.2, we can also express the following, less straightforward, relations between triangles (we will use the derived predicates listed in Section 3.2 to make the presented formulas as clear as possible):

• The predicate $\mathbf{EqualArea}(\Delta_p, \Delta_q)$ can be expressed in FO[**Δ**].

We can express in FO[**Δ**] the fact that that two triangles have equal area. Indeed, the fact that two triangles have equal area is affine invariant (affinities preserve ratios of areas) and can be expressed in FO[+,×,<,0,1] (two triangles have the same area if the determinants of the coordinates of their points have the same absolute value). We can obtain the FO[**Δ**] expression by translating the FO[+,x,<,0,1]-formula into a FO[**Between**]-formula (see [8]) and then translating this last formula into FO[**Δ**]using the translation explained in Lemma 2. We will not give this formula here as it is very long.

From **EqualArea**, we can also derive the predicate **Bigger:**

• The expression $\mathbf{Bigger}(\Delta_p, \Delta_q)$ stands for the formula

$$(\exists \Delta_r)((\Delta_r \neq \Delta_p) \wedge \mathbf{PartOf}(\Delta_r, \Delta_p) \wedge \mathbf{EqualArea}(\Delta_r, \Delta_q)).$$

• The predicate $\mathbf{CenterOM}(\Delta_p, \Delta_q)$ expresses that a (triangle degenerated into a) point is the center of mass of another triangle. The center of mass of a triangle is the intersection of the medians of that triangle. The median of a triangle connects a corner point with the middle of the opposite side. Therefore, the median of a triangle divides the triangle into two parts (also triangles) of equal area. The expression

$$\begin{aligned}&\mathbf{Point}(\Delta_p) \wedge \mathbf{PartOf}(\Delta_p, \Delta_q) \wedge \mathbf{RealTriangle}(\Delta_q) \wedge \\ &\quad (\exists \Delta_{r_1}, \Delta_{r_2}, \Delta_{r_3}, \Delta_{r_4}, \Delta_{r_5}, \Delta_{r_6})\Big(\bigwedge_{i,j=1, i\neq j}^{6} (\Delta_{r_i} \neq \Delta_{r_j}) \wedge \\ &\qquad \mathbf{EqPart}(\Delta_{r_1}, \Delta_{r_2}, \Delta_q) \wedge \mathbf{EqPart}(\Delta_{r_3}, \Delta_{r_4}, \Delta_q) \wedge \\ &\qquad\quad \mathbf{EqPart}(\Delta_{r_5}, \Delta_{r_6}, \Delta_q) \wedge \bigwedge_{i=1}^{6} \mathbf{PartOf}(\Delta_p, \Delta_{r_i})\Big).\end{aligned}$$

defines $\mathbf{CenterOM}(\Delta_p, \Delta_q)$.

Here, $\mathbf{EqPart}(\Delta_q, \Delta_r, \Delta_p)$ is expressed by the formula

$$\begin{aligned}&\mathbf{EqualArea}(\Delta_q, \Delta_r) \wedge (\Delta_q \neq \Delta_r) \wedge \mathbf{PartOf}(\Delta_q, \Delta_p) \wedge \\ &\quad \mathbf{PartOf}(\Delta_r, \Delta_p) \wedge \mathbf{LineAdj}(\Delta_q, \Delta_r) \wedge \neg(\exists \Delta_s)(\mathbf{PartOf}(\Delta_s, \Delta_p) \wedge \\ &\qquad \mathbf{RealTriangle}(\Delta_s)) \wedge \neg\mathbf{PartOf}(\Delta_s, \Delta_q) \wedge \neg\mathbf{PartOf}(\Delta_s, \Delta_r).\end{aligned}$$

See also Figure 3 for an illustration of the construction of the center of mass of a triangle.

5.2 Examples of Tqueries on Tbases

Consider a Tbase that contains information about butterflies and flowers. The unary Trelation *Butter flyB* contains all regions where some butterfly *B* is spotted. The unary Trelation *PlantP* contains all regions where some specific plant

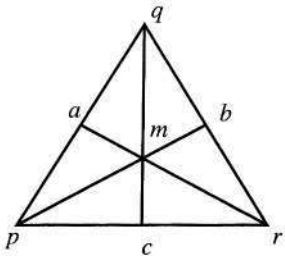

Fig. 3. The center of mass m of triangle *pqr* is the intersection of the triangles *pqb, pbr, qpc, qrc, rqa* and *rpa,* where the segments [*pb*], [*qc*] and [*ra*] are medians of the triangle *pqr.*

P grows. We also have a Trelation *Rural,* containing rural regions. It is known in biology that each butterfly appears close to some specific plant, as caterpillars only eat the leaves of their favorite plant. Suppose that it is also investigated that butterflies like to live in rural areas.

• $\mathbf{Q}_A$ **: Are all butterflies** *B* **spotted in regions where the plant** *P* **grows?** This query can be used to see if it is possible that a butterfly was spotted in a certain region. The query $\mathbf{Q}_A()$ can be expressed by the formula

$$\neg((\exists \Delta_p, \Delta_q)(ButterflyB(\Delta_p) \wedge \mathbf{RealTriangle}(\Delta_q) \wedge \mathbf{PartOf}(\Delta_q, \Delta_p) \wedge \neg((\exists \Delta_r)(PlantP(\Delta_r) \wedge \mathbf{PartOf}(\Delta_q, \Delta_r))))).$$

• $\mathbf{Q}_B$: **Give the region(s) where we have to search if we want to see butterfly** *B.* The query $\mathbf{Q}_B(\Delta_p)$ can be expressed by the formula

$$(\exists \Delta_q, \Delta_r)(PlantP(\Delta_q) \wedge Rural(\Delta_r) \wedge \mathbf{PartOf}(\Delta_p, \Delta_q) \wedge \mathbf{PartOf}(\Delta_p, \Delta_r)).$$

• $\mathbf{Q}_C$ **: Give the region inside the convex hull of the search region for butterfly** *B.* It is much more convenient to search a convex region than having to deal with a very irregularly shaped region.

We first express how to test whether the region is convex ($\mathbf{Q}'_C$), this will help understand the formula that computes the convex hull. The query $\mathbf{Q}'_C()$ can be expressed by the formula

$$(\forall \Delta_{p_1}, \Delta_{p_2}, \Delta_{p_3}, \Delta_p)((\bigwedge_{i=1}^{3} \mathbf{Point}(\Delta_{p_i}) \wedge \bigwedge_{i=1}^{3} \mathbf{Q}_B(\Delta_{p_i}) \wedge \mathbf{CornerP}(\Delta_{p_1}, \Delta_{p_2}, \Delta_{p_3}, \Delta_p)) \Rightarrow (\mathbf{Q}_B(\Delta_p))).$$

The expression

$$(\exists \Delta_{p_1}, \Delta_{p_2}, \Delta_{p_3}, \Delta_{q_1}, \Delta_{q_2}, \Delta_{q_3})(\bigwedge_{i=1}^{3} \mathbf{Point}(\Delta_{p_i}) \wedge$$

$$\bigwedge_{i=1}^{3} \mathbf{PartOf}(\Delta_{p_i}, \Delta_{q_i}) \wedge \bigwedge_{i=1}^{3} \mathbf{Q}_B(\Delta_{q_i}) \wedge \mathbf{CornerP}(\Delta_{p_1}, \Delta_{p_2}, \Delta_{p_3}, \Delta_p))$$

hence defines the query $\mathbf{Q}_C(\Delta_p)$.

- $\mathbf{Q}_D :=$ **Give the boundary of Region** $\mathbf{Q}_C$**.**

The boundary is computed by considering all segments that are on the boundary. Such segments are line-adjacent to a triangle inside the region, but there is no triangle on the other side of the segment within the region. Only the maximal such segments are returned. See also Figure 4 for an illustration. The query $\mathbf{Q}_D(\Delta_p)$ can be expressed by the formula

$$\begin{aligned}
&\mathbf{Q}_C(\Delta_p) \wedge \mathbf{Line}(\Delta_p) \wedge (\exists \Delta_q)(\mathbf{LineAdj}(\Delta_p, \Delta_q) \wedge \mathbf{Q}_C(\Delta_q) \wedge \\
&\mathbf{RealTriangle}(\Delta_q) \wedge \neg(\exists \Delta_r)(\mathbf{LineAdj}(\Delta_r, \Delta_q) \wedge \mathbf{LineAdj}(\Delta_r, \Delta_p) \wedge \\
&\quad \mathbf{Q}_C(\Delta_r) \wedge \mathbf{RealTriangle}(\Delta_r))) \wedge \neg(\exists \Delta_l)(\mathbf{Q}_C(\Delta_l) \wedge \mathbf{Line}(\Delta_l) \wedge \\
&\quad\quad \mathbf{PartOf}(\Delta_p, \Delta_l) \wedge (\exists \Delta_s)(\mathbf{LineAdj}(\Delta_l, \Delta_s) \wedge \mathbf{Q}_C(\Delta_s) \wedge \\
&\quad\quad\quad \mathbf{RealTriangle}(\Delta_r) \neg(\exists \Delta_t)(\mathbf{LineAdj}(\Delta_t, \Delta_s) \wedge \\
&\quad\quad\quad\quad \mathbf{LineAdj}(\Delta_t, \Delta_l) \wedge \mathbf{Q}_C(\Delta_t) \wedge \mathbf{RealTriangle}(\Delta_t)))).
\end{aligned}$$

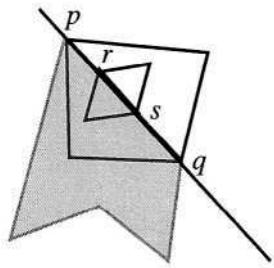

Fig. 4. The line segment *pq* is a boundary segment of the grey polygon. The segment *rs* belongs to the boundary but is not a maximal boundary segment.

The first two queries ask for relations between regions that can be expressed by the so-called 9-intersection model [6]. This model defines a relation between two regions by investigating the intersections between their boundaries, interiors and exteriors. As the boundary, interior and exterior of a region can be expressed in FO[+,×,<,0,1], and are affine invariant concepts, all relations that can be expressed by the 9-intersection model, can be expressed in FO[$\mathbf{\Delta}$].

6 Safety of Tqueries

The definition of Tbases allows triangle relations to represent an infinite set of triangles. In practice, however, spatial databases will contain only finite sets

of triangles. The question that arises naturally is whether the language $\mathrm{FO}[\boldsymbol{\Delta}]$ returns a finite output on finite input. The answer is no (see Example 3 below). Moreover, safety of FO[+,×,<,0,1]-queries is undecidable in general [1], so we cannot decide *a priori* whether a Tquery will return a finite output or not.

The following example illustrates the fact that the language $\mathrm{FO}[\boldsymbol{\Delta}]$ does not necessarily return finite output on finite input.

Example 3. Let R be a relation containing a finite number of triangles. Consider the following queries:

- Q_1 : **Give all triangles that are part of some triangle of** R.
  The query $Q_1(\Delta_p)$ is expressed by the formula

$$(\exists \Delta_q)(R(\Delta_q) \wedge \mathbf{PartOf}(\Delta_p, \Delta_q)).$$

- Q_2 : **Give all triangles that intersect some triangle of** R.
  The query $Q_2(\Delta_p)$ can be expressed by the formula

$$(\exists \Delta_q)(R(\Delta_q) \wedge \mathbf{Intersect}(\Delta_p, \Delta_q)).$$

- Q_3 : **Give all the corner points of all triangles of** R.
  The query $Q_3(\Delta_p)$ can be expressed by the formula

$$(\exists \Delta_q, \Delta_r, \Delta_s)(R(\Delta_q) \wedge (\mathbf{CornerP}(\Delta_p, \Delta_r, \Delta_s, \Delta_q) \vee \mathbf{CornerP}(\Delta_r, \Delta_p, \Delta_s, \Delta_q) \vee \mathbf{CornerP}(\Delta_s, \Delta_r, \Delta_p, \Delta_q))).$$

The queries Q_1 and Q_2 return an infinite set of triangles. The query Q_3 returns a finite number of triangles on a finite input. □

As we cannot decide whether a Tquery will return a finite result, we turn to the question of determining whether the result itself is finite or not. The answer is affirmative:

Proposition 1. It is decidable whether a Trelation is finite. Moreover, there exists a $\mathrm{FO}[\boldsymbol{\Delta}]$ query that decides finiteness of a Trelation. □

Proof sketch. A Trelation of arity k corresponds to a semi-algebraic set in $\mathbf{R}^{6k}$. A Trelation is finite if and only if the corresponding semi-algebraic set contains a finite number of points. It is well known that there exists a FO[+,×,<,0,1]-formula deciding whether a semi-algebraic set contains a finite number of points. The fact that a semi-algebraic set contains a finite number of points is affine-invariant. From the fact that the property is affine-invariant and expressible in FO[+,×,<,0,1], it follows that we can, in $\mathrm{FO}[\boldsymbol{\Delta}]$, express the formula that decides whether a Trelation is finite or not (using Theorem 1). □

We do have a means of deciding whether a Trelation is finite, but it seems this requirement is too restrictive. The reason is the following. Intuitively, when one thinks of a unary Trelation or the result of a unary query, one pictures a set of triangles as all points in the plane belonging to one of the triangles, i.e., the *drawing* of this set in the plane.

Definition 13. Let Δ_p be a triangle. The *drawing* of Δ_p is the union of all points in the plane that are part of Δ_p.

Let R be a Trelation of arity one. The *drawing* $\mathcal{D}^R$ of R is the two-dimensional figure that is the union of the drawings of all triangles in R. □

Remark that different Trelations can have the same drawing. Therefore, it seems natural to extend the strict notion of finiteness of a Trelation to the existence of a finite Trelation having the same drawing. For example, query Q_1 from Example 3 seems to be a query we would like to allow, as the union of all triangles that are part of a given triangle, is that given triangle itself. Query Q_2 clearly returns an infinite set of triangles, that is not even finitely representable. This is the query we would not want to allow.

Fortunately, given the output of a unary query, we can determine whether its drawing can be represented as a finite union of triangles.

For the remainder of this text, we restrict Trelations and Tqueries to be unary. It is not clear immediately if it would make sense to define drawings on relations or queries with an arity greater than one. For example, consider a binary relation containing only one tuple of line-adjacent non-degenerated triangles. If we draw this relation, we would like to draw both triangles participating in the relation. This gives the same result as the drawing of a unary relation containing two tuples. So the drawing "wipes out" the relationship between the triangles.

Proposition 2. It is decidable whether the output of a unary Tquery on a particular finite input can be represented as a finite union of triangles. Furthermore, the fact that the output of the Tquery can be represented as a finite union of triangles can be expressed in $\mathrm{FO}[\mathbf{\Delta}]$. □

Proof sketch. It is clear that if the drawing of a Trelation can be represented as a finite union of triangles, it can be represented by a FO[+,×,<,0,1]-formula using only polynomials of degree at most one. Indeed, each set that can be described using polynomials of at most degree one, is called a semi-linear set. It is well-known that the bounded semi-linear sets are the same as finite unions of bounded polytopes.

So, if we can check whether the drawing of a (possibly infinite) set of triangles is bounded and can be represented using polynomials of degree at most one, we know that the set can be represented by a finite number of triangles.

Checking whether the drawing of a Trelation R is bounded can be done easily in $\mathrm{FO}[\mathbf{\Delta}]$: $\mathbf{IsBounded}() := (\exists \Delta_p)((\forall \Delta_q)(R(\Delta_q) \Rightarrow \mathbf{PartOf}(\Delta_q, \Delta_p)))$.

Also, we can decide whether a two-dimensional semi-algebraic set can be represented using polynomials of degree at most k, for any natural number k [14].

There exists a FO[+,×,<,0,1] formula deciding this [14][4]. It is clear that the drawing of a unary Trelation is a semi-algebraic set.

From the facts that computing the drawing of a Trelation is an affine-generic query that can be expressed in FO[+,×,<,0,1], that checking whether a Trelation has a bounded drawing can be expressed in FO[$\mathbf{\Delta}$], that there exists a FO[+,×,<,0,1]-formula deciding whether the drawing of a Trelation can be expressed by polynomials of degree at most one can be done in FO[+,×,<,0,1], and, finally, that the fact that the drawing of a Trelation can be expressed by polynomials of degree at most one is affine-invariant, we conclude that we can decide whether a Trelation has a finite representation, and we can construct a FO[$\mathbf{\Delta}$]-formula deciding this. □

We now show that, if the drawing of the output of a Tquery is representable as a finite set of triangles, we can compute such a finite representation in FO[$\mathbf{\Delta}$].

First, we describe briefly an affine-invariant triangulation method for 2-dimensional bounded semi-linear figures. This triangulation was introduced by Haesevoets, Kuijpers and Revesz [9]. A triangulation of a set S is a finite set of triangles, each pair of triangles having at most one corner point or line segment in common, whose union equals S. A triangulation method is affine-invariant if for each pair of figures S and S' such that $S' = \alpha(S)$, for some affinity α of $\mathbf{R}^2$, a triangle Δ_p is an element of the triangulation of S if and only if the triangle $\Delta_q = \alpha(\Delta_p)$ is an element of the triangulation of S'.

We now give the outline of the affine-invariant triangulation algorithm.

Algorithm 1.

Input: A unary Trelation S that can be represented as a finite union of triangles.
Output: An affine-invariant triangulation of S.

- Step-1: Compute the boundary B_S of S. B_S is a finite set of line segments and points.
- Step-2: Compute the set of carriers[5] for all segments of B_S. Those carriers partition S into a finite union of open convex polygons, points and open line segments. All closures of line segments that do not form a side of one of the convex polygons, together with all points that are not a corner point of one of the convex polygons are returned as degenerated triangles. Remark that we can return the closures of the line segments as S originally is a union of closed triangles, closed line segments and points.
- Step-3: Triangulate each polygon that is not a triangle already, by connecting each corner point to the center of mass[6] of the polygon. Return all triangles composed of the center of mass and two consecutive corner points. If the polygon is a triangle already, output it as is. Remark that, as S originally is a union of closed triangles, closed line segments and points, the closures of all convex polygons found in Step-2 belong to S, and we can return closed triangles.

[4] It is not expressible for arbitrary dimensions, only for two-dimensional semi-algebraic sets.

[5] The *carrier* of a line segment is the line through that line segment.

[6] The center of mass of the convex polygon P with corner points $p_1 = (x_1, y_1), p_2 = (x_2, y_2), \ldots, p_k = (x_k, y_k)$ is the point with coordinates $(\frac{x_1+x_2+\cdots+x_k}{k}, \frac{y_1+y_2+\cdots+y_k}{k})$.

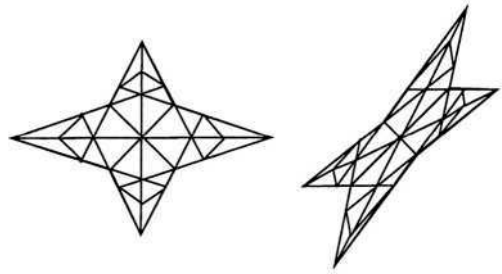

Fig. 5. A star-shaped figure (left) and an affine image (right) are triangulated using Algorithm 1.

Figure 5 gives an example of a triangulation of a star-like figure and an affine image of the same image.

We conjecture that the triangulation as described in Algorithm 1 cannot be expressed in FO[$\mathbf{\Delta}$]. We conjecture that the center of mass of a polygon, which is affine-invariant, cannot be expressed in FO[+,×,<,0,1] [12], and therefore, also not in FO[$\mathbf{\Delta}$].

Conjecture 1. Let P be a convex polygon with corner points $p_1, p_2, \ldots, p_k$, for $k > 3$. The center of mass of P cannot be expressed in FO[+,×,<,0,1]. □

If we relax the requirement to have a *partition* of the original figure into triangles to having a *finite union* of triangles representing the figure, we can change the last step of Algorithm 1 as follows:

Algorithm 2.

Input: A unary Trelation S that can be represented as a finite union of triangles.
Output: An affine-invariant finite representation of S.

- Step-1: Compute the boundary B_S of S. B_S is a finite set of line segments and points.
- Step-2: Compute the set of carriers for all segments of B_S. Those carriers partition S into a finite union of open convex polygons, points and open line segments. All closures of line segments that do not form a side of one of the convex polygons, together with all points that are not a corner point of one of the convex polygons are returned as degenerated triangles. Remark that we can return the closures of the line segments as S originally is a union of closed triangles, closed line segments and points.
- Step-3: For each polygon that is not a triangle already, output the finite set of triangles that connect three distinct corner points of the polygon. If the polygon is a triangle already, output it as is.

We will call the procedure described in this algorithm as the *affine finite representation* of *S,* or, abbreviated, *AFR*(*S*).

Figure 6 illustrates the difference between Step-3 of Algorithm 1 and Step-3 of Algorithm 2.

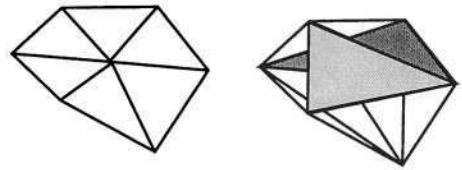

Fig. 6. The hexagon on the left is triangulated using its center of mass. The hexagon on the right is represented as a finite union of triangles, connecting three corner points. Two such triangles are shaded in (light resp., dark) grey.

Proposition 3. We can compute, given a unary Trelation R that can be represented as a finite union of triangles, the set $AFR(\mathcal{D}^{\mathcal{R}})$ in FO[Δ]. □

Proof sketch. The affine finite representation is an affine invariant.
We only have to prove this for Step-3 of Algorithm 2. The rest is shown in [9].

Let $P = p_1, p_2, \ldots, p_k$ be a polygon with k corner points, for some integer number $k > 3$. Let α be an affinity of the plane. It is clear that, for each triangle $(p_h, p_i, p_j)(h \neq i \neq j; 1 \leq h, i, j \leq k)$ of the triangle representation of P, the triangle $\alpha(p_h, p_i, p_j) = (\alpha(p_h), \alpha(p_i), \alpha(p_j))$ is an element of the triangle representation of $\alpha(P)$.

The affine finite representation is computable in FO[+,×,<,0,1].
In FO[+,×,<,0,1], it is possible to compute the boundary of a semi-linear set (Step-1). It is also possible to compute the carriers of all boundary segments, and their intersection points (Step-2). The affine finite representation of the convex polygons defined by the carriers is computed by returning all triangles connecting three intersection points of carriers, such that their interior is not intersected by some carrier (Step-3).

From the fact that the triangle representation is affine invariant and computable in FO[+,×,<,0,1], it follows that it is computable in FO[Δ]. □

7 Conclusion

In this article, we introduced a new triangle-based query language, denoted FO[Δ]. The use of triangles instead of points or real numbers is motivated by the spatial (spatio-temporal) practice, where data is often represented as a collection of (moving) triangles.

We showed that our query language has the same expressiveness as the affine-invariant FO[+,×,<,0,1]-queries on triangle databases. We did this by showing that our language is sound and complete for the FO[**Between**]-queries on triangle databases.

Afterwards, we gave several examples to illustrate the expressiveness of the triangle-based language and the ease of use of manipulating triangles.

We then turned to the notion of safety. We showed that, although we cannot decide whether a particular Tquery returns a finite output given a finite input, we can decide whether the output is finite. We also extended this finiteness to

the more intuitive notion of sets that have a finite representation. We proved that we can decide whether the output of a query has a finite representation and compute such a finite representation in $\mathrm{FO}[\Delta]$.

Besides the intuitive manipulation of spatial data represented as a collection of triangles, another motivation for this language is that it can serve as a first step towards a natural query language for spatio-temporal data that are collections of *moving triangles.*

Geerts, Haesevoets and Kuijpers [7] already proposed point-based languages for several classes of spatio-temporal queries. The data model used there represented a moving two-dimensional object as a collection of points in three-dimensional space. There exist however, data models that represent spatio-temporal data as a collection of moving objects (see for example [4, 5]), which is more natural. Hence, a *moving triangle*-based language with the same expressiveness as the spatio-temporal point languages mentioned above would be much more useful in practice.

Acknowledgements

The author would like to thank Bart Kuijpers for giving useful suggestions, in specific for suggesting Conjecture 1, and the anonymous reviewers for their detailed comments that helped to improve this paper.

References

1. M. Benedikt and L. Libkin. Safe constraint queries. *SIAM Journal on Computing,* 29(5):1652–1682, 2000.
2. M. D. Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry.* Springer-Verlag, 2000.
3. J. Bochnak, M. Coste, and M. Roy. *Géométrie Algébrique Réelle*. Springer-Verlag, Berlin, 1987.
4. C. X. Chen and C. Zaniolo. SQLST: A spatio-temporal data model and query language. In A. H. F. Laender, S. W. Liddle, and V. C. Storey, editors, *Conceptual Modeling, 19th International Conference on Conceptual Modeling (ER'00),* volume 1920 of *Lecture Notes in Computer Science,* pages 96–111. Springer-Verlag, 2000.
5. J. Chomicki, S. Haesevoets, B. Kuijpers, and P. Revesz. Classes of spatiotemporal objects and their closure properties. *Annals of Mathematics and Artificial Intelligence,* (39):431–461, 2003.
6. M. Egenhofer and J. Herring. A mathematical framework for the definition of topological relationships. In K. Brassel and H. Kishimoto, editors, *Proceedings of the Fourth International Symposium on Spatial Data Handling,* pages 803–813, 1990.
7. F. Geerts, S. Haesevoets, and B. Kuijpers. A theory of spatio-temporal database queries. In G. Ghelli and G. Grahne, editors, *Database Programming Languages, 8th International Workshop, DBPL 2001,* volume 2397 of *Lecture Notes in Computer Science,* pages 198–212. Springer-Verlag, 2002.
8. M. Gyssens, J. Van den Bussche, and D. Van Gucht. Complete geometric query languages. *Journal of Computer and System Sciences,* 58(3):483–511, 1999.

9. S. Haesevoets, B. Kuijpers, and P. Revesz. Efficient indexing of image databases using novel affine-invariant color and shape features. *Manuscript,* 2003.
10. M. Hagedoorn and R. C. Veldkamp. Reliable and efficient pattern matching using an affine invariant metric. *International Journal of Computer Vision,* 31:203–225, 1999.
11. D. Huttenlocher, G. Klauderman, and W. Rucklidge. Comparing images using the hausdorff distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence,* 15:850–863, 1998.
12. B. Kuijpers. Personal communication, 2003.
13. B. Kuijpers, J. Paredaens, and J. Van den Bussche. On topological elementary equivalence of spatial databases. In F. Afrati and P. Kolaitis, editors, *Proceedings of the 6th International Conference on Database Theory (ICDT'97),* volume 1186 of *Lecture Notes in Computer Science,* pages 432–446. Springer-Verlag, 1997.
14. B. Kuijpers and M. Smits. On expressing topological connectivity in spatial datalog. In V. Gaede, A. Brodsky, O. Günther, D. Srivastava, V. Vianu, and M. Wallace, editors, *Constraint Databases and Their Applications (CDB'97),* volume 1191 of *Lecture Notes in Computer Science,* pages 116–133. Springer-Verlag, 1997.
15. Y. Lamdan, J. Schwartz, and H. Wolfson. Affine-invariant model-based object recognition. *IEEE Journal of Robotics and Automation*, 6:578–589, 1990.
16. R. Laurini and D. Thompson. *Fundamentals of Spatial Information Systems.* Number 37 in APIC Series. Academic Press, 1992.
17. G. Nielson. A characterization of an affine invariant triangulation. In G. Farin, H. Hagen, and H. Noltemeier, editors, *Geometric Modelling, Computing Supplementum 8,* pages 191–210, 1993.
18. C. Papadimitriou, D. Suciu, and V. Vianu. Topological queries in spatial databases. In *Proceedings of the 15th ACM Symposium on Principles of Database Systems,* pages 81–92. ACM Press, 1996.
19. J. Paredaens, J. V. den Bussche, and D. V. Gucht. Towards a theory of spatial database queries. In *Proceedings of the 13th ACM Symposium on Principles of Database Systems,* pages 279–288, New York, 1994. ACM Press.
20. J. Paredaens, G. Kuper, and L. Libkin, editors. *Constraint databases.* Springer-Verlag, 2000.
21. P. Revesz. *Introduction to Constraint Databases.* Springer-Verlag, 2002.
22. L. Roberts. Machine perception of three-dimensional solids. *J. T. Tippet, editor, Optical and Electro-optical Information Processing,* 1965.

Appendix

Proof of Lemma 1

Remark that a triangle variable Δ_p is naturally translated into a triple of points $(\hat{p}_1, \hat{p}_2, \hat{p}_3)$. The order in which the points are taken is unimportant. Some of the points may possibly coincide.

Proof sketch. We prove this by induction on the structure of $\mathrm{FO}[\mathbf{\Delta}]$-formulas. We only give the translations here. The correctness of each translation can be proved straightforwardly. We omit the correctness proofs for the translations because of space restrictions.

1. Atomic formulas

We first show that all atomic formulas of FO[$\mathbf{\Delta}$] can be expressed in the language FO[**Between**]. Those atomic formulas are equality constraints on triangle variables, the predicates of $\mathbf{\Delta}$ applied to point variables, and the relation names from σ applied to triangle variables.

- **The translation of** $\Delta_p = \Delta_q$ is

$$\bigvee_{\sigma(1,2,3)=(i_1,i_2,i_3),\sigma\in\mathcal{S}_3} (\hat{p}_1 = \hat{q}_{i_1}, \hat{p}_2 = \hat{q}_{i_2}, \hat{p}_3 = \hat{q}_{i_3}),$$

where $\mathcal{S}_3$ is the set of all permutations of {1,2,3}.

- **The translation of PartOf**(Δ_p, Δ_q) is

$$\bigwedge_{i=1}^{3} \mathbf{InTriangle}(\hat{p}_i, \hat{q}_1, \hat{q}_2, \hat{q}_3),$$

where the definition of **InTriangle** is (see Figure 1 for an illustration):

$$\mathbf{InTriangle}(\hat{s}, \hat{p}, \hat{q}, \hat{r}) := (\exists \hat{t})(\mathbf{Between}(\hat{p}, \hat{t}, \hat{q}) \wedge \mathbf{Between}(\hat{t}, \hat{s}, \hat{r})).$$

- **The translation of Line**(Δ_p) is

$$\bigvee_{i=1}^{3} ((\hat{p}_i = \hat{p}_{(i+1)\mathrm{Mod}3}) \wedge (\hat{p}_i \neq \hat{p}_{(i+2)\mathrm{Mod}3})).$$

- **The translation of** $R(\Delta_{p_1}, \Delta_{p_2}, \ldots, \Delta_{p_k})$ is

$$\hat{R}(\hat{p}_1^1, \hat{p}_2^1, \hat{p}_3^1, \hat{p}_1^2, \hat{p}_2^2, \hat{p}_3^2, \ldots, \hat{p}_1^k, \hat{p}_2^k, \hat{p}_3^k),$$

where $\hat{R}$ is a geometric relation of arity $3k$, obtained by the canonical bijection between $((\mathbf{R}^2)^3)^k$ and $(\mathbf{R}^2)^{3k}$.

We now show that all composed formulas of FO[$\mathbf{\Delta}$] can be expressed in FO[**Between**].

2. Composed formulas

Atomic formulas of FO[$\mathbf{\Delta}$] are composed by the operations conjunction ($\wedge$), disjunction ($\vee$) and negation ($\neg$), and existential quantifiers ($\exists$).

We assume that we already translated the FO[$\mathbf{\Delta}$]-formulas φ and ψ into the FO[**Between**]-formulas $\hat{\varphi}$ and $\hat{\psi}$.

The translation of respectively $\varphi \wedge \psi$, $\varphi \vee \psi$, $\neg\varphi$ and $(\exists\Delta_p)\varphi$ are:

$$\hat{\varphi} \wedge \hat{\psi}\ ,\ \hat{\varphi} \vee \hat{\psi}\ ,\ \neg\hat{\varphi} \text{ and } (\exists \hat{p}_1, \hat{p}_2, \hat{p}_3)\hat{\varphi}.$$

□

Proof of Lemma 2

Proof sketch. We have to prove that we can translate every Tquery in the language FO[**Between**] into a query in the language FO[$\mathbf{\Delta}$]. We only give the translations here. The correctness of each translation can be proved straightforwardly. We omit the correctness proofs for the translations because of space restrictions.

We do this by induction on the formulas of FO[**Between**]. First, we show that we can translate each atomic formula of FO[$\mathbf{\Delta}$] into FO[**Between**].

1. Translation of the atomic formulas of FO[Between] into FO[$\mathbf{\Delta}$]
The atomic formulas of the language FO[$\mathbf{\Delta}$] are equality constraints on triangle variables, the predicates of $\mathbf{\Delta}$ applied to point variables, and the relation names from σ applied to triangle variables

- **The translation of a point variable** $\hat{p}$ is the triangle variable Δ_p,
  and we add the condition $\mathbf{Point}(\Delta_p)$ to the beginning of the translation of the sub formula where the variable occurs first.
- **The translation of** $\hat{p} = \hat{q}$ is $\Delta_p =_\Delta \Delta_q$.
- **The translation of** $\hat{R}(\hat{p}_1^1, \hat{p}_2^1, \hat{p}_3^1, \ldots, \hat{p}_1^k, \hat{p}_2^k, \hat{p}_3^k)$ is

$$(\exists \Delta_{p^1}, \ldots, \Delta_{p^k})(R(\Delta_{p^1}, \ldots, \Delta_{p^k}) \wedge \bigwedge_{i=1}^{k} \mathbf{CornerP}(\Delta_{p_1^i}, \Delta_{p_2^i}, \Delta_{p_3^i}, \Delta_{p^i})).$$

As before, R is obtained from $\hat{R}$ by the canonical bijection between $(\mathbf{R}^2)^{3k}$ and $((\mathbf{R}^2)^3)^k$.

- **The translation of** $\mathbf{Between}(\hat{p}, \hat{q}, \hat{r})$ is

$$\begin{aligned}\mathbf{Point}(\Delta_p) \wedge \mathbf{Point}(\Delta_q) \wedge \mathbf{Point}(\Delta_r) \wedge (\exists \Delta_s, \Delta_t)(\mathbf{Line}(\Delta_s) \wedge \\ \mathbf{Line}(\Delta_t) \wedge \mathbf{SameCarrier}(\Delta_s, \Delta_t) \wedge \mathbf{PartOf}(\Delta_p, \Delta_s) \wedge \\ \mathbf{PartOf}(\Delta_q, \Delta_s) \wedge \neg\mathbf{PartOf}(\Delta_r, \Delta_s) \wedge \\ \mathbf{PartOf}(\Delta_q, \Delta_t) \wedge \mathbf{PartOf}(\Delta_r, \Delta_t) \wedge \neg\mathbf{PartOf}(\Delta_p, \Delta_t)).\end{aligned}$$

This formula expresses first that Δ_p, Δ_q, and Δ_r are collinear by stating that the segments with respective corner points Δ_p and Δ_q, and Δ_q and Δ_r should have the same carrier. Furthermore, Δ_q is only between Δ_p and Δ_r if the segment with corner points Δ_p and Δ_q does not contain Δ_r, and, in an analog way, the line segment with corner points Δ_q and Δ_r does not contain Δ_p.

2. The translation of composed formulas
Atomic formulas of FO[**Between**] are composed by the operations conjunction ($\wedge$), disjunction ($\vee$) and negation ($\neg$), and existential quantifiers ($\exists$).

We assume that we already translated the FO[**Between**]-formulas φ and ψ into the FO[$\mathbf{\Delta}$]-formulas φ_Δ and ψ_Δ.

The translation of respectively $\varphi \wedge \psi$, $\varphi \vee \psi$, $\neg\varphi$ and $(\exists \hat{p})\varphi$ are:

$$\varphi_\Delta \wedge \psi_\Delta \ , \ \varphi_\Delta \vee \psi_\Delta \ , \ \neg\varphi_\Delta \text{ and } (\exists \Delta_p)(\mathbf{Point}(\Delta_p) \wedge \varphi_\Delta).$$

In general, the translation of a FO[**Between**] query

$$\hat{Q}(\hat{p}_1^1, \hat{p}_1^2, \hat{p}_1^3, \ldots, \hat{p}_k^1, \hat{p}_k^2, \hat{p}_k^3)$$

is the query $Q(\Delta_{p_1}, \ldots, \Delta_{p_k})$, expressed by the formula

$$(\exists \Delta_{p_1^1}, \Delta_{p_1^2}, \Delta_{p_1^3}, \ldots, \Delta_{p_k^1}, \Delta_{p_k^2}, \Delta_{p_k^3})(\bigwedge_{i=1}^{k} \mathbf{CornerP}(\Delta_{p_i^1}, \Delta_{p_i^2}, \Delta_{p_i^3}, \Delta_{p_i}) \wedge \overline{Q}(\Delta_{p_1^1}, \Delta_{p_1^2}, \Delta_{p_1^3}, \ldots, \Delta_{p_k^1}, \Delta_{p_k^2}, \Delta_{p_k^3})),$$

where $\overline{Q}$ is the translation of $\hat{Q}$ as explained above. □

Applying Constraint Databases in the Determination of Potential Minimal Conflicts to Polynomial Model-Based Diagnosis

Maria Teresa Gómez López, Rafael Ceballos Guerrero,
Rafael Martínez Gasca, and Carmelo del Valle Sevilla

Departamento de Lenguajes y Sistemas Informáticos,
Escuela Técnica Superior de Ingeniería Informática, Universidad de Sevilla, Spain
{mayte,ceballos,gasca,carmelo}@lsi.us.es

Abstract. Model-based Diagnosis allows the identification of the parts which fail in a system. The models are based on the knowledge of the system to diagnose, and may be represented by constraints associated to the components. The variables of these constraints can be observable or non-observable, depending on the situation of the sensors. In order to obtain the potential minimal diagnosis in a system, an important issue is related to finding out the potential minimal conflicts in an efficient way. We consider that Constraint Databases represent an excellent option in order to solve this problem in complex systems.

In this work we have used a novel logical architecture of Constraint Databases which has allowed obtaining these potential conflicts by means of the corresponding queries. Moreover, we have considered Gröbner Bases as a projection operator to obtain the potential minimal conflicts of a system. The first results obtained on this work, which are shown in a heat exchangers example, have been very promising.

1 Introduction

In the industrial production, the faults produced in components and processes can cause undesirable stops and damage in the systems with the consequent cost rise and production decrease. It is also necessary to take into account that these faults can produce a negative impact on the environment, what has to be avoided. For this reason, in order to keep the systems within the desired security, production and reliability levels, mechanisms which allow the detection and diagnosis of the faults produced in the systems have to be developed.

Diagnosis allows us to identify the failures in a system. Most approaches proposed in the last decade use models (FDI and DX approaches) based on the knowledge of the system to diagnose. These models can be formally well structured or can be known by means of an expert's experience and data of the system or process. Sometimes, a combination of both types of knowledge can also be presented. In the area of DX, the first work related to diagnosis was presented with the objective of identifying faults in the component systems based on the

B. Kuijpers and P. Revesz (Eds.): CDB 2004, LNCS 3074, pp. 74–87, 2004.

structure and its behavior [3]. The first implementations to perform diagnosis were DART [11] and GDE [5], which used different inference mechanisms to detect possible faults.

Diagnosis formalization was presented in [20,6], where a general theory was proposed for the problem of explaining the discrepancies between the observed and correct behaviors that the mechanisms subject to the diagnosis process (logical-based diagnosis) have. Based on this, most DX approaches for components characterize the diagnosis of a system as a collection of potential minimal sets of failing components which explain the observed behaviors (symptoms). The importance of having a good model in DX can be deduced from this. This kind of diagnosis can correctly diagnose the important parts of the component systems that are failing. An exhaustive revision of the approaches about diagnosis task automation can be found in [7], and a discussion about model-based diagnosis for applications can be consulted in [2].

The models centered on mechanisms describe the systems by means of input-output relations. Model-based diagnosis is considered by a pair {SD,OBS} where SD is the system description and OBS is a set of values of the observable variables. In complex systems, it gets data from several observations, equations of the system description and contexts at an enormous rate. In engineering applications, it is often overlooked the storage of these data and query processing.

In this paper, we present how some issues concerning Constraint Database can be the improvement of the efficiency in some phases of the model-based diagnosis. In concrete we tackle the determination of potential minimal conflicts of a system. A conflict is a set of assumptions where at least one must be false. The assumptions are about behavioral modes of components. GDE coupled with an ATMS [4] as inference engine uses previously discovered conflicts to constrain the search in the candidate space. In this approach, conflicts are identified in the constraint propagation process through recording dependencies of predicted values given the system description and the observations. A conflict is minimal, if none of its subsets is a conflict. The main disadvantage of using it is a large number of possible conflicts 2^n, where n is the number of components. In the previous years, this problem has been an active area of research, in order to find a minimal conflict [18] or the finding all potential minimal conflicts, using CS-Tree [15] or using preprocessing, independence and incrementally features of initial model [9]. In the DX community there are a lot of works in this sense as the calculation the minimal chains which can be evaluated, the minimal models which can also be evaluated [19] and symbolic processing algorithms (Gröbner bases) of the initial model [10]. The use of Gröbner bases in the FDI community were also proposed in previous works [8].

Our work presents a novel methodology that uses Constraint Databases technology for the determination of potential minimal conflicts with the object of promoting the advantages of this technology and its uses in real problem of industrial diagnosis. There are some positive features to many model-based applications that can be exploited by a Constraint Database. First the easy representation of the component-based model of a technical system by means of a

set of algebraic constraints and during the search in the space of possible contexts to model-based diagnosis, the constraints of the different contexts differ in only a few constraints. This similarity can be exploited by incremental solving techniques.

In our paper, the model is stored in a Polynomial Constraint Database and a preprocessing step reduces in a significant way the number of possible contexts to treat by means of symbolic techniques. This technique is based on the algorithm for computing Gröbner bases given by Buchberger in [1]. This algorithm is our projection operator and we will use it to eliminate the non-observable variables of the constraints in the different contexts.

Moreover the relational model is not able to represent the scientific and engineering data. For this reason we consider Constraint Databases [17,16,21, 12,13] as a good tool for our reasoning processing.

Our paper has been organized as follows: firstly, we present definitions and notation which allow us to formalize the subsequent operations. In sections 3 we show an example to prove our solution and use a Constraint Database architecture to treat the information. Afterwards, we show the obtained results when we applied our methodology to the set of heat exchangers system example. Finally we present our conclusions and future works in this research line.

2 Definitions and Notation

The definitions and notation used are based on the concepts developed in the diagnosis community, based itself on the logic (DX) and on redundancies (FDI). The objective is that the synergy of both approaches will produce diagnosis results which are as representative as possible of what is happening in the system in the shortest time. As it has already been mentioned, model-based diagnosis requires a system model. This time, we will only deal with the case which has a model of system constraints that derives from its own structure, and which has links between components (structural model) and the behavior of each model component. With this model and with the idea of formalizing the diagnosis process, the definitions and notation used in the development of this work need to be exposed.

Definition 1. The System Polynomial Model (SPM): It can be defined as a finite set of polynomial equality constraints P which determine the system behavior. This is done by means of the relation between the system non-observable variables (Vnob) and the observable variables (Vob) which are directly obtained from sensors that are supposed to work correctly. Then, the following tuple for a system polynomial model is obtained SPM (P,Vob,Vnob).

Definition 2. Diagnosis Problem (DP): It can be defined by means of a tuple formed by a System Polynomial Model and an Observational Model. The result of this problem will be a set of elements that belong to the set of the system faults which reflects, in a minimal way, the information of the possible failing components DP(SPM,OM). In this work we are not going to study the Ob-

servational Model, we only propose an improvement of the systems polynomial model.

Definition 3. Context Network: It is a graph formed by all the elements of the context set of the system according to the way proposed by ATMS [4], In our work this context network will be enriched with the Context Analytical Redundancy Constraints.

Definition 4. Context Analytical Redundancy Constraint (CARC): Set of constraints derived from SPM and in such a way that only the observed variables are related. In this work, we are only dealing with the models defined by polynomial equality constraints. In these constraints, their truth value can be evaluated from the observed variables of the system through the corresponding monitorization.

3 A Heat Exchangers System: A Case Study

To explain our methodology, we have been applied it to a system, like the one shown in figure 1 from [10] and [14]. In this system, consisting of six heat exchangers, three flows f_i come in at different temperatures t_i. The behaviors of the system is described by polynomial constraints coming from three different kinds of balances:

$\sum_i f_i = 0$: mass balance at each node,
$\sum_i f_i * t_i = 0$: thermal balance at each node,
$\sum_{in} f_i * t_i - \sum_{out} f_j * t_j = 0$: enthalpic balance for each heat exchanger.

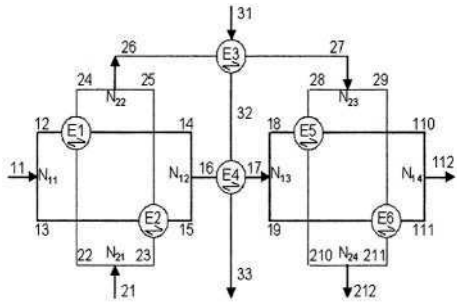

Fig. 1. System of heat exchangers

The resulting system, thus, consists of 34 equations and 54 variables, from which 28 are observable: f_{11}, f_{12}, f_{13}, f_{16}, f_{17}, f_{18}, f_{19}, f_{112}, f_{21}, f_{26}, f_{27}, f_{212}, f_{31}, f_{33}, t_{11}, t_{12}, t_{13}, t_{16}, t_{17}, t_{18}, t_{19}, t_{112}, t_{21}, t_{26}, t_{27}, t_{212}, t_{31} and t_{33}. There is no a direct measure of the rest of the variables. This defines three different subsystems, each one formed by two exchangers: {E1, E2}, {E3, E4} and {E5, E6}. Each of the six exchangers and each of the eight nodes of the system are considered as components whose correct functioning have to be verified.

Table 1. System Polynomial Model of the System of Heat Exchangers

C.	Constraints	C.	Constraints
N_{11}	f_{11}-f_{12}-f_{13}	E_1	f_{12}-f_{14}
	$f_{11}\cdot t_{11}$-$f_{12}\cdot t_{12}$-$f_{13}\cdot t_{13}$		f_{22}-f_{24}
N_{12}	f_{14}+f_{15}-f_{16}		$f_{12}\cdot t_{12}$-$f_{14}\cdot t_{14}$+$f_{22}\cdot t_{22}$-$f_{24}\cdot t_{24}$
	$f_{14}\cdot t_{14}$+$f_{15}\cdot t_{15}$-$f_{16}\cdot t_{16}$	E_2	f_{13}-f_{15}
N_{13}	f_{17}-f_{18}-f_{19}		f_{23}-f_{25}
	$f_{17}\cdot t_{17}$-$f_{18}\cdot t_{18}$-$f_{19}\cdot t_{19}$		$f_{13}\cdot t_{13}$-$f_{15}\cdot t_{15}$+$f_{23}\cdot t_{23}$-$f_{25}\cdot t_{25}$
N_{14}	f_{110}+f_{111}-f_{112}	E_3	f_{26}-f_{27}
	$f_{110}\cdot t_{110}$+$f_{111}\cdot t_{111}$-$f_{112}\cdot t_{112}$		f_{31}-f_{32}
N_{21}	f_{21}-f_{22}-f_{23}		$f_{26}\cdot t_{26}$-$f_{27}\cdot t_{27}$+$f_{31}\cdot t_{31}$-$f_{32}\cdot t_{32}$
	$f_{21}\cdot t_{21}$-$f_{22}\cdot t_{22}$-$f_{23}\cdot t_{23}$	E_4	f_{16}-f_{17}
N_{22}	f_{24}+f_{25}-f_{26}		f_{32}-f_{33}
	$f_{24}\cdot t_{24}$+$f_{25}\cdot t_{25}$-$f_{26}\cdot t_{26}$		$f_{16}\cdot t_{16}$-$f_{17}\cdot t_{17}$+$f_{32}\cdot t_{32}$-$f_{33}\cdot t_{33}$
N_{23}	f_{27}-f_{28}-f_{29}	E_5	f_{18}-f_{110}
	$f_{27}\cdot t_{27}$-$f_{28}\cdot t_{28}$-$f_{29}\cdot t_{29}$		f_{28}-f_{210}
N_{24}	f_{210}+f_{211}-f_{212}		$f_{18}\cdot t_{18}$-$f_{110}\cdot t_{110}$+$f_{28}\cdot t_{28}$-$f_{210}\cdot t_{210}$
	$f_{210}\cdot t_{210}$+$f_{211}\cdot t_{211}$-$f_{212}\cdot t_{212}$	E_6	f_{19}-f_{111}
			f_{29}-f_{211}
			$f_{19}\cdot t_{19}$-$f_{111}\cdot t_{111}$+$f_{29}\cdot t_{29}$-$f_{211}\cdot t_{211}$
V_{ob}	$=f_{11},f_{12},f_{13},f_{16},f_{17},f_{18},f_{19},f_{21},f_{26},f_{27},f_{112},f_{212},f_{31},f_{33},t_{11},t_{12},t_{13},t_{16},t_{17},$ $t_{18},t_{19},t_{112},t_{21},t_{26},t_{27},t_{212},t_{31},t_{33}$		
V_{nob}	$=f_{14},f_{15},f_{21},f_{22},f_{23},f_{24},f_{25},f_{28},f_{29},f_{210},f_{211},f_{110},f_{111},f_{32},t_{14},t_{15},t_{110},t_{111},t_{22},$ $t_{23},t_{24},t_{25},t_{28},t_{29},t_{210},t_{211},t_{32}$		

For the example presented in figure 1, the SPM is represented in Table 1. The context network for this example is too large to be shown, but in order to clarify this concept, we present only one subsystem in the figure 2. This subsystem includes the components $\{N_{12}, N_{21}, N_{22}, E_1, E_2\}$.

4 Computing Potential Minimal Conflicts

The model which reflects the system structure and behavior contains the constraints that link the system inputs and outputs. Many times some intermediate variables are not observable and do not allow to determine whether there are faults in the components in a direct way. Then our idea is to produce an equivalent constraints model which has the same solution as the original one, but without non-observable variables. In order to transform the polynomial constraints set of the system, we apply a function that obtains Gröbner bases.

To study the potential minimal conflicts with common techniques, we must rebuild the full problem, if our system changes because we add or delete some components. To solve this problem we will store information in a Constraint Database. Also, if we do not use a Constraint Database and the execution of the program diagnosis fails while being executed, we must reexecute the full problem. It is because there is not any partial information stored. If we save the

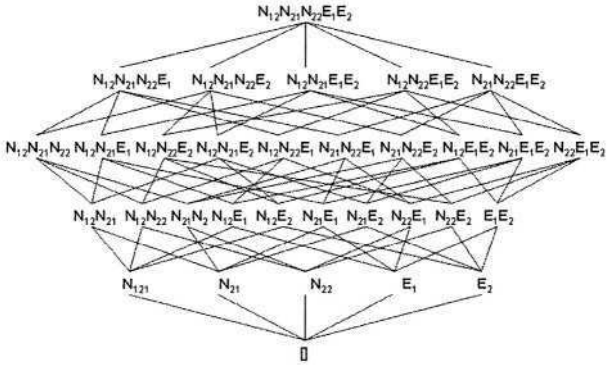

Fig. 2. Context Network for a subsystem of the heat exchangers example. The components included are $\{N_{12}, N_{21}, N_{22}, E_1, E_2\}$

information in a database step by step, we can continue the process with the obtained information. Constraint Databases allows us to use the power of SQL standard in order to query the database and to obtain the necessary information.

In the following two subsections we explain the most important properties about Gröbner bases and the Constraint Database architecture. In the last section we propose an algorithm to obtain the potential minimal conflicts based on Gröbner bases and Constraint Databases.

4.1 Gröbner Bases

Gröbner bases theory is the origin of many symbolic algorithms used to manipulate multiple variable polynomials. The algorithm used for the polynomial equations system is based on the ideas proposed in [1]. It is a generalization of Gauss' elimination of multivariable lineal equations and of Euclides' algorithm for one-variable polynomial equations. Gröbner bases has better computational properties than the original system. We can be said that it is very easy to determine if the system can be solved or not.

The main idea is to transform the polynomial constraint set into a standard form for the resolution of problems. Having the set of equality polynomial constraints of the form P=0, Gröbner bases produce an equivalent system G=0 which has the same solution as the original one, but it is generally easier to solve.

For our work, we have a function which is called GröbnerBasis. This function calculates Gröbner bases by means of a finite set of polynomial equations (SPM) and a set of observable and non-observable variables.

This function obtains the different contexts of a particular model and allows the building of the context network. The signature of GröbnerBasis function looks like this:

GröbnerBasis({Polynomials},{Observable Variables},{Non-observable Variables})

Let us consider, for instance, the context represented by the following components $\{N_{12}E1E2\}$. Then, *GröbnerBasis* function takes the parameters:

$\text{GröbnerBasis}(\{\text{polynomialsOf}(N_{12}, E1, E2)\}, \{f_{16}, f_{12}, f_{13}, t_{16}, t_{12}, t_{13}\},$
$\{f_{14}, f_{15}, f_{13}, f_{15}, t_{14}, t_{15}, t_{13}, t_{15}\})$

The result would be the following system of polynomial constraints:

$\{f_{12} + f_{13} - f_{16} = 0\}$

4.2 Constraint Database Architecture

One of the difficulties in diagnosing a system is handling the information. In this paper we propose to store the information in a relational database. It will allow us to store partial results.
Also, it is important to highlight the power of the query language in a database. Using a database helps to improve the diagnosis and to access data.

First of all, we are going to explain the database architecture, and how we store the information. It is shown in figure 3. The semantics of these storage elements are explaining in the following items:

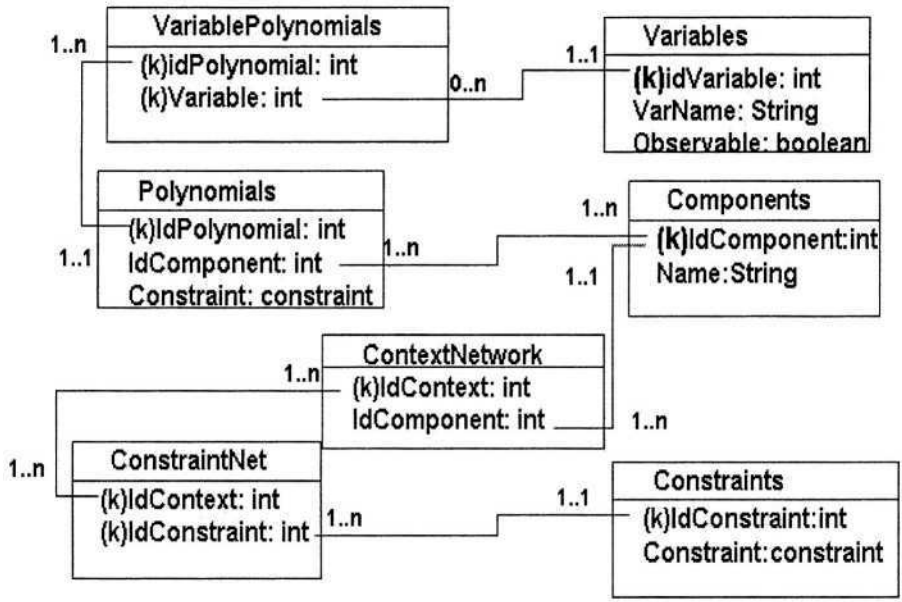

Fig. 3. Constraint Database Architecture (k: Primary Key)

Explanations about Constraint Database tables (figure):

1. **Components:** This table contains the names and identifiers of the components which make up the system.

2. **Polynomials:** This table contains the different behaviors of the components. The components can have more than one polynomial associated with them, like the example of the figure 1. An example of this table for our problem is shown in table 2.

Table 2. Polynomial table

IdPolynomial	IdComponent	Constraint
0	0	$f_{11} - f_{12} - f_{13}$
1	0	$f_{11} * t_{11} - f_{12} * t_{12} - f_{13} * t_{13}$
2	1	$f_{14} + f_{15} - f_{16}$
3	1	$f_{14} * t_{14} + f_{15} * t_{15} - f_{16} * t_{16}$

3. **ContextNetwork:** This table represents all the relations that the process must study to obtain the minimal possible conflict context. The table has potentially 2^n combination of elements, where n is the number of components that constitute the system.
4. **Variables:** Here all the variables which take part in the system are stored, observable as well as non-observable. An example of it is shown in the table 3.

Table 3. Variables Table

IdVariable	VarName	Observable
25	t_{212}	Yes
26	t_{31}	Yes
27	t_{33}	Yes
28	f_{14}	No
29	f_{15}	No
30	f_{110}	No

5. **VariablePolynomials:** This table represents the variables of each polynomial. This table is important because to obtain Gröbner bases we need to use the observable and non-observable variables of each polynomial.
6. **Constraints:** All the constraints are stored in this table. These constraints have only observable variables. We will fill in this table with the GröbnerBasis results.
7. **ConstraintNet:**This table relates each context and the constraints related to them.

4.3 Constraint Databases Approach

If we call GröbnerBasis for all contexts (2^{14} times) to obtain the conflictive contexts will take 4 days and 2 hours, and it will obtain 64 constraints. Some of these 64 constraints are redundant because some of them are linear combination of others. For that reason, we must detect these combinations and reduce them, as it is done in [10].

We propose to use Constraint Databases and the power of SQL standard to query the database and obtain the data. It helps us to improve the time of obtaining the possible minimal conflictive contexts, and therefore obtain just the necessary information.

Using the constraints databases and the SQL query language, we are proposing a new way to improve the first solution. To improve the solution mentioned above, we have avoided these two disadvantages of it. The disadvantages are the handling of the data and the great amount of calls to GröbnerBasis function. To improve it we use the algorithm of the figure 4.

```
1 for(i := 1 to i=numComponents)
2   j := 1
3   boolean promising:=true
4   while(promising AND j + i ≤numComponents)
5     if(IsAnObservableContext(i))
6       AddContext(i)
7       UpdateTables(i)
8       promising:=false
9     else
10        Set contexts=ObtainContexs(i, j))
11        for each c of contexts:
12          if (RelevantContext(c))
13            AddContext(c)
14            CallGröbner(c)
15          endif
16        endforeach
17    endif
18    j := j + 1
19  endwhile
20 endfor
```

Fig. 4. Pseudocode of the reduction algorithm

This algorithm offers a way to improve the obtaining of minimal possible conflictive contexts without creating all the contexts nor calling to GröbnerBasis function all the times.

In order to explain the algorithm, we need to add two new definitions.

Definition 5. *Observable Context*: It is a context without non-observable variables. It means that we do not have to eliminate any variables. An example of an observable context is $\{N_{11}\}$.

Definition 6. *Relevant Context*: It is a context whose components have, at least, one polynomial whose non-observable variables also are in other components of the same context. If we call GröbnerBasis in other cases, we will not obtain any important results, because it is not possible to eliminate all non-observable variables of at least one polynomial of all the components of the context:

C is a relevant context if
$\mathbf{C} \equiv \bigcup_i \{c_i\} \mid \forall\, c_i \in \mathbf{C} \cdot \exists\, p_i \in c_i$
$\mid \forall\, \mathbf{x} \in \mathbf{NonObsVar}(p_i) \cdot \mathbf{x} \in \mathbf{C} \setminus c_i$
c_i being a component, p_i a polynomial and $\mathbf{NonObsVar}(p_i)$ the set of non-observable variable of p_i

Methods of the reduction algorithm:

- *IsAnObservableContext():* This method returns true if the context has only one component and this component does not have any polynomial with non-observable variables.
- *AddContext(Context c):* Add to table ContextNetwork the context c.
- *UpdateTables(Context c):* Input all the polynomial constraints of the context c, in the tables ConstraintNet and Constraint. Here it is not necessary to call GröbnerBasis because the polynomials do not have non-observable variables, and Gröbner method does not have any non-observable variables to eliminate.
- *ObtainContext(Integer i, Integer j):* This method returns all the contexts with j components which have the component i in them.
- *RelevantContext(Context c):* Returns true if the context c is a relevant context.

In section 4.3 it is explained how many calls to Gröbner are necessary to study the full system. But if the system studies all the combinations, the cost of time is very high. Moreover, if we try to execute all the contexts, we will get redundant information that should be handled later. We propose a way to reduce the use of Gröbner, and only use it when it is necessary, when the results are interesting and non-redundant.

Example 1:
Context: N_{22}, E1 and E2
In this case the component E1 does not have any polynomial with all non-observable variable couple with other component.

Example 2:
Context: N_{12}, E1 and E2
In this case all the components have any polynomial with all its non-observable variable couple with other component.

To implement previous idea, we propose an interesting query to know which are the non-observable variables of a *polynomial constraint* which are also in the same *context* but in distinct component.

```
SELECT DISTINCT v.VARNAME
FROM VARIABLES v, VARIABLES v2, VARIABLEPOLYNOMIALS cv,
   VARIABLEPOLYNOMIALS cv2, CONTEXTNETWORK rc,
   CONTEXTNETWORK rc2, POLYNOMIALS c, POLYNOMIALS c2
```

```
WHERE c.ID=polynomial AND
   c.IDCOMPONENT=rc.IDCOMPONENT AND rc.ID=context
   AND c.ID=cv.ID AND cv.VARIABLE=v.IDVARIABLE AND
   v.OBSERVABLE=false AND c.ID<>c2.ID AND AND
   rc2.ID=rc.ID c2.IDCOMPONENT=rc2.IDCOMPONENT AND
   c2.ID=cv2.ID AND cv.VARIABLE=cv2.VARIABLE AND
   c.IDCOMPONENT<>c2.IDCOMPONENT
```

And with the next query, we will know which are the non-observable variables of a polynomial constraint:

```
SELECT v.VARNAME
FROM VARIABLES v, VARIABLEPOLYNOMIAL cv
WHERE cv.ID=polynomial AND v.IDVARIABLE=cv.VARIABLE
   AND v.OBSERVABLE=false
```

Comparing both results, we will know if all the non-observable variables of a polynomial are in another component.

With these two queries we can check if it is a relevant context.

- *CallGröbner()*;We build GröbnerBasis function with information of the tables ContextNetwork, Polynomials, VariablePolynomial and Components. After the execution of the function we will pick the results up. If the solution is not in Constraint table, we will add it. Finally, we will store the constraint and the context related in the table CostraintNet.

Explanation of the reduction algorithm:
Our reduction algorithm studies each possible context before it is created and calls GröbnerBasis. In line 1 each component is selected, and in line 4 it will be possible to study all the contexts which have the component i in their j components. In line 5 the algorithm studies if the context has only one component $(j == 1)$, and if it is an observable component. In this case we do not have to study all the possible contexts with the component i, because they will not be relevant. To avoid this useless computation (example 3), we use the boolean variable *promising*. Let us see the following example:

Example 3:

Context: $N_{11}, E3$ *and* $E4$

Really this context is constituted by two subcontexts: $\{N_{11}\}$ *and* $\{E3, E4\}$. *Thereby is not necessary to study the full context because we will obtain redundant information*

With this algorithm, we only create 43 contexts as an alternative to 2^{14} of the first solution. With our solution we only call GröbnerBasis 41 times, because the contexts $\{N_{11}\}$ and $\{N_{13}\}$ are observable contexts. Our reduction algorithm spends 12 minutes and 27 seconds and obtains 17 CARCs which are shown on table 4.

Table 4. *Minimal conflict constraints for the system (CARCs)*

1	$f_{11} - f_{12} - f_{13}$
2	$f_{11} * t_{11} - f_{12} * t_{12} - f_{13} * t_{13}$
3	$-f_{12} - f_{13} + f_{16}$
4	$f_{21} - f_{26}$
5	$-(f_{13} * t_{12}) + f_{16} * t_{12} + f_{13} * t_{13} - f_{16} * t_{16} + f_{26} * t_{21} - f_{26} * t_{26}$
6	$-f_{112} + f_{18} + f_{19}$
7	$f_{212} - f_{27}$
8	$f_{31} - f_{33}$
9	$f_{26} - f_{27}$
10	$f_{21} - f_{27}$
11	$-(f_{17} * t_{16}) + f_{17} * t_{17} - f_{27} * t_{26} + f_{27} * t_{27} - f_{33} * t_{31} + f_{33} * t_{33}$
12	$f_{16} - f_{17}$
13	$f_{13} * t_{12} - f_{17} * t_{12} - f_{13} * t_{13} + f_{17} * t_{17} - f_{27} * t_{21} + f_{27} * t_{27} - f_{33} * t_{31} + f_{33} * t_{33}$
14	$-f_{12} - f_{13} + f_{17}$
15	$f_{18} * t_{112} + f_{19} * t_{112} - f_{18} * t_{18} - f_{19} * t_{19} + f_{27} * t_{212} - f_{27} * t_{27}$
16	$f_{17} - f_{18} - f_{19}$
17	$f_{17} * t_{17} - f_{18} * t_{18} - f_{19} * t_{19}$

4.4 Context Analytical Redundancy Constraint

In this section we will study, with the help of SQL and Java^{TM} languages, if there are any contexts whose polynomial constraints have already been in another context with less components. One example is shown in the figure 5. In this case we can eliminate the context $\{N_{14}, E3, E4, E5, E6\}$, because all their constraints are in contexts with less components.

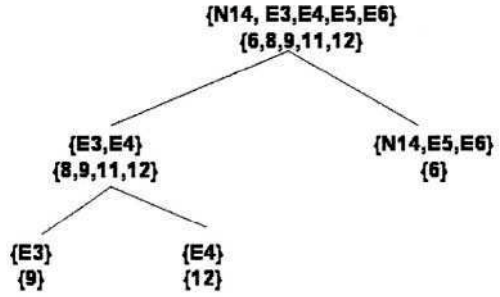

Fig. 5. Elimination of Redundancies

With this elimination technique, we have deleted 31 contexts with redundant constraints. Thereby we have 12 contexts and 17 constraints, as we show in the figure 6. To improve this result we are studying to use a module to eliminate some linear combination.

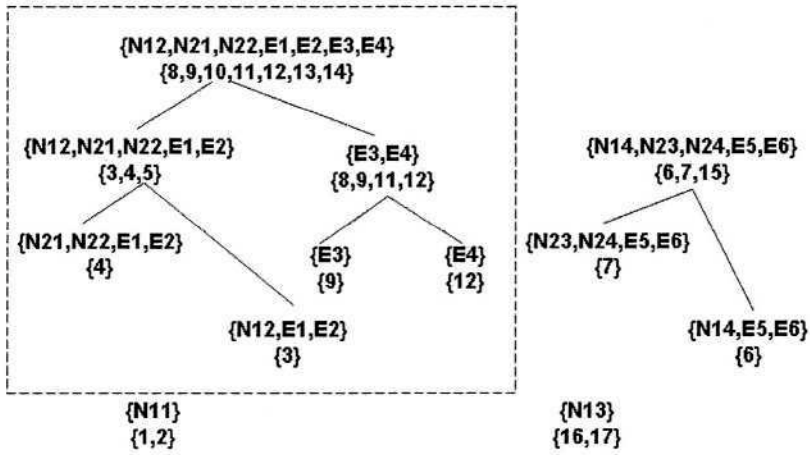

Fig. 6. Minimal Context Network for the system

5 Conclusions and Future Works

In this paper we have proposed a database architecture to store polynomial constraints in a Constraints Database. Using SQL standard and Java languages is possible to obtain and handle the polynomial constraint of the model.The proposed reduction algorithm has several advantages. One is the computational improvement in the calculation of the potential minimal set conflicts. Another advantage is to make the process automatic. Finally it is important highlight the power of SQL that offers to store and get information in Constraint Databases in a easy way.

As future works we want to improve our methodology dividing the system into several subsystems. If the system is divided, it can be studied separately. It is interesting in systems which have observable variables which allow to know the problem by parts. The problem of the heat exchangers is an example of this because it can be divided into five different parts. Also, we are considering to study how the minimal context network changes when some polynomial constraints change, and to look for techniques to avoid restudying all the system.

Anyway, there is a great field to study the creation of systems with components located in different Constraint Databases.

Acknowledgements

This work has been partially funded for the Ministerio de Ciencia y Tecnología of Spain (DPI2003-07146-C02-01) and European Regional Development Fund. (ERDF/FEDER).

References

1. B. Buchberger. Gröbner bases: An algorithmic method in polynomial ideal theory. *In Multidimensional Systems Theory, N. K. Bose, ed., D. Reidel Publishing Co.*, pages 184–232, 1985.
2. L. Console and O. Dressler. Model-based diagnosis in the real world: Lessons learned and challenges remaining. *In Proceedings IJCAI' 99,* pages 1393–1400, 1999.
3. R. Davis. Diagnostic reasoning based on structure and behavior. *In Artificial Intelligence*, 24:347–410, 1984.
4. J. de Kleer. An assumption-based truth maintenance system. *Artificial Intelligence 28(2)*, pages 127–161, 1986.
5. J. de Kleer and Williams B.C. Diagnosing multiple faults. *Artificial Intelligence*, 1987.
6. J. de Kleer, A. Mackworth, and R. Reiter. Characterizing diagnoses and systems. *In Artificial Intelligence*, 56(2-3):197–222, 1992.
7. E. Frisk. The consistency-based approach to automated diagnosis of devices. *In Brewka(ed). Principles of Knowledge Representation, CSLI*, 1996.
8. E. Frisk. Residual generator design for non-linear, polynomial systems - a gröbner basis approach. *In Proc. IFAC Safeprocess 2000, Budapest Hungary*, pages 979–984, 2000.
9. M. Garcia de la Banda, P. Stuckey, and J. Wazny. Finding all minimal unsatisfiable subsets proc. *Of the 5th ACM Sigplan Internacional*, 2003.
10. R.M Gasca, C Del Valle, R. Ceballos, and M. Toro. An integration of fdi and dx approaches to polynomial models. *DX*, 2003.
11. M. Genesereth. The use of design descriptions in automated diagnosis. *Artificial Intelligence*, 24:411–436, 1984.
12. D.Q. Goldin. Constraint query algebras. *Doctorate thesis*, 1997.
13. D.Q. Goldin and P.C. Kanellakis. Constraint query languages. *Constraint Query Algebras Constraints Journal*, 1996.
14. C. Guernez. Fault detection and isolation on non linear polynomial system. *5th IMACS Word Congress on Scientific, Computation, Modelling and Applied Mathematics*, 1997.
15. B. Han and S. Lee. Deriving minimal conflict sets by cs-trees with mark set in diagnosis from first principles. *IEEE Transcations on Systems, man and cybernetics*, 29(2), 1999.
16. P. C. Kanellakis, G. M. Kuper, and P.Z. Revesz. Constraint query languages. *Symposium on Principles of Database Systems*, pages 299–313, 1990.
17. G. Kuper, L. Libkin, and J. Paredaes. *Constraint Databases.* Springer, 1998.
18. Mauss and Mugur Tatar. Computing minimal conflicts for rich constraint languages jakob. *DX,* 2002.
19. J. B. Pulido. Posibles conflictos como alternativa al registro de dependencias en lnea para el diagnstico de sistemas continuos. *Ph. Doctoral Dissertation. Universidad de Valladolid*, 2000.
20. R. Reiter. A theory of diagnosis from first principles. *In Artificial Intelligence*, 32(1):57–96, 1987.
21. P. Revesz. *Introduction to Constraint Databases.* Springer, 2002.

Constraint Database Solutions to the Genome Map Assembly Problem

Viswanathan Ramanathan and Peter Revesz

Department of Computer Science and Engineering
University of Nebraska-Lincoln, Lincoln, NE 68588, USA
{ramanath,revesz}@cse.unl.edu

Abstract. Long DNA sequences have to be cut using restriction enzymes into small fragments whose lengths and/or nucleotide sequences can be analyzed by currently available technology. Cutting multiple copies of the same long DNA sequence using different restriction enzymes yields many fragments with overlaps that allow the fragments to be assembled into the order as they appear on the original DNA sequence. This basic idea allows several NP-complete abstractions of the genome map assembly problem. However, it is not obvious which variation is computationally the best in practice. By extensive computer experiments, we show that in the average case the running time of a constraint automata solution of the *big-bag matching* abstraction increases linearly, while the running time of a greedy search solution of the *shortest common superstring in an overlap multigraph* abstraction increases exponentially with the size of real genome input data. Hence the first abstraction is much more efficient computationally for very large real genomes.

1 Introduction

Constraint databases can be useful in the area of genomics. In particular, we can solve the genome map assembly problem, described in detail in Section 2, very efficiently using constraint databases and the closely related concept of constraint automata.

Not only is a solution possible in the constraint database framework, but it turns out to be a much more efficient solution than earlier proposals based on greedy search for a common shortest superstring in an overlap multigraph, which is also described in Section 2.

In this paper we propose a modification of the big-bag matching abstraction, which is then solved using a constraint automaton. The earlier constraint automaton solution was a pure fingerprints-based method. Our modification is to disambiguate the fingerprint data during a search by comparing the nucleotide sequences of the ambiguous data. Hence ours is a hybrid method between pure fingerprinting and pure string comparison methods.

By extensive computer experiments we show that the running time of the modified constraint automaton solution grows essentially linearly while the running time of the other solution grows approximately exponentially with the size of the input genome data. This is surprising, because both abstractions are

B. Kuijpers and P. Revesz (Eds.): CDB 2004, LNCS 3074, pp. 88–111, 2004.

```
CATCGATCTCGGGAGGGATCCATTATCGATTCCCGGGCTCGGGGGATCCT
TCCATCGATGGGCCCGAGGCGGATCCCTACTATCGATCCCGGGGGGATCC
TTAATTCTCGAGAAGGCCTATCGATCAAGGATCCTATCGATCCCGAGTCC
CGGGAT
```

Fig. 1. A genome sequence part of Lambda bacterium.

known to be NP-complete, hence the two algorithms were hard to distinguish theoretically.

This paper is organized as follows. Section 2 describes some basic concepts and related work. Section 3 explains the proposed modified constraint automaton solution and its implementation. Section 4 presents the results and the statistical analysis of the experiments conducted. Section 5 discusses the use of constraint databases in this project. Finally, Section 6 gives some conclusions and directions for future work.

2 Basic Concepts and Related Work

Unfortunately, there have been only few papers exploring the connection between genomics and constraint databases, because of the need to understand a significant number of concepts in both areas. Hence we will start with a brief review of the key definitions and related work.

Genome. The genome of an organism is its set of *chromosomes,* containing all of its *genes* and the associated *deoxyribonucleic acid* (DNA) [4]. DNA is a double-stranded helix consisting of nucleotides. Each nucleotide has three parts: a sugar molecule (S), a phosphate group (P), and a structure called a nitrogenous base (A, C, G, and T). The DNA is built on the repeating sugar-phosphate units. The sugars are molecules of deoxyribose from which DNA receives its name. Joined to each deoxyribose is one of the four possible nitrogenous bases: *Adenine* (A), *Cytosine* (C), *Guanine* (G), and *Thymine* (T). These bases carry the genetic information, so the words *nucleotide* and *base* are often used interchangeably. Since the DNA is double-stranded, the bases on one strand are linked to the bases on the other strand to form a base pair. Adenine always pairs with Thymine and Guanine always pairs with Cytosine. The length of a DNA is measured in terms of number of base pairs.

For example, Fig. 1 shows a part of one strand of the Lambda bacterium DNA [5]. Its length is 156 base pairs.

Restriction Enzymes. *Restriction enzymes* are precise molecular scalpels that allow a scientist to manipulate DNA segments. They recognize specific base sequences in double-helical DNA and cut, at specific places, both strands of a duplex containing the recognized sequences [1]. They are indispensable tools for analyzing chromosome structure, sequencing very long DNA molecules, isolating genes, and creating new DNA molecules that can be cloned.

Many restriction enzymes recognize specific sequences of four to eight base pairs. Their names consist of a three-letter abbreviation (e.g., *Eco* for *Escherichia*

coli) followed by a strain designation (if needed) and a roman numeral (if more than one restriction enzyme from the same strain has been identified) (eg. `EcoRI`).

For example, we can apply the restriction enzyme `ClaI`, which always cuts the genome at each site where `AT^CGAT` appears with the wedge symbol showing where the cut will take place. Then we obtain the subsequence shown in Fig. 2.

A1: CAT
A2: CGATCTCGGGAGGGATCCATTAT
A3: CGATTCCCGGGCTCGGGGGATCCTTCCAT
A4: CGATGGGCCCGAGGCGGATCCCTACTAT
A5: CGATCCCGGGGGGGATCCTTAATTCTCGAGAAGGCCTAT
A6: CGATCAAGGATCCTAT
A7: CGATCCCGAGTCCCGGGAT

Fig. 2. Subsequences of the Lambda bacterium sequence.

If we use another restriction enzyme `BamHI`, which cuts at sites `G^GATCC`, then we obtain the subsequences shown in Fig. 3.

B1: CATCGATCTCGGGAGG
B2: GATCCATTATCGATTCCCGGGCTCGGGG
B3: GATCCTTCCATCGATGGGCCCGAGGCG
B4: GATCCCTACTATCGATCCCGGGGG
B5: GATCCTTAATTCTCGAGAAGGCCTATCGATCAAG
B6: GATCCTATCGATCCCGAGTCCCGGGAT

Fig. 3. Another set of subsequences of the same Lambda bacterium sequence.

Genome Sequencing and Mapping. *Genome sequencing* is the process of finding the order of DNA nucleotides, or bases, in a genome [4]. On the other hand, *genome mapping* is the process of finding the approximate position of landmarks (specific subsequences, often genes) in a genome without getting into the details of the actual sequence. A genome map is less detailed than a genome sequence [4]. A sequence spells out the order of every nucleotide in the genome, while a map simply identifies the order of the specified subsequences in the genome. Nevertheless, the two are closely related concepts, because by sequencing each landmark of a genome map, we can get a genome sequence.

For example, suppose we consider each of the seven subsequences in Fig. 2 as a landmark. Then a genome map would be the following:

$$A1\ A2\ A3\ A4\ A5\ A6\ A7$$

Sometimes we can have only a partial genome map. Suppose that only $A2$ and $A5$ are known landmarks. Then a partial genome map would be the following:

$$-\ A2\ -\ -\ A5\ -\ -$$

where the dashes are unknown regions on the genome map.

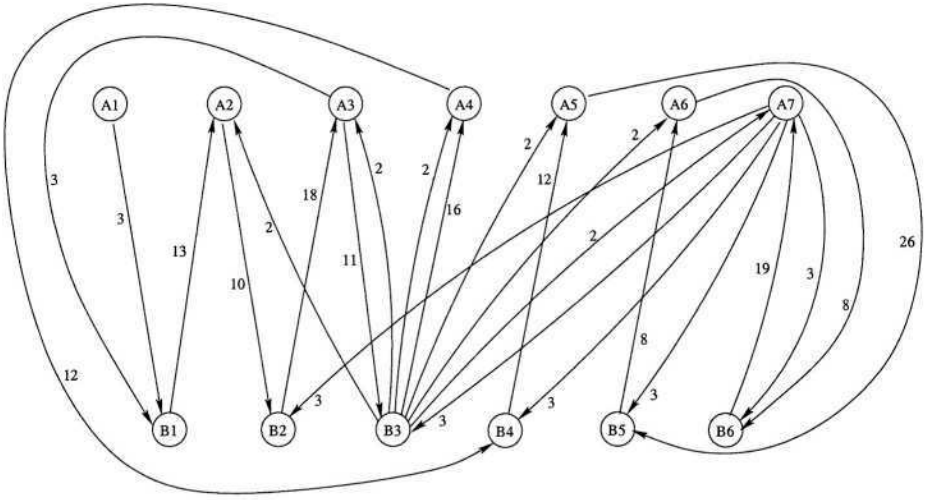

Fig. 4. The overlap multigraph for F.

Genome Map Assembly Problem. DNA sequences are huge, having a length of around 200-300 million base pairs for many animals and plants. But current sequencing machines can not handle DNA sequences of length more than a couple of thousand base pairs. So the DNA sequences have to be cut into small subsequences using restriction enzymes. After the application of a restriction enzyme the subsequences obtained are floating a solution. Hence we no longer know their original order. So, once the subsequences are sequenced and analyzed, they have to be arranged and assembled to obtain the original sequence. This process is called *Genome Map Assembly* [8]. The problem of executing the Genome Map Assembly process is called the *Genome Map Assembly Problem* (GMAP) [20].

Overlap Multigraph. The *overlap multigraph* method [25] is one way of doing the genome map assembly. This method is usually illustrated by a graph in which each node is one of the subsequences and each directed edge from node A to node B has a label $k \geq 0$ if and only if the last k nucleotides of A and the first k nucleotides of B are the same.

For example, let F be the union of the As and Bs in Figs. 2 and 3. Then part of the overlap multigraph of F is shown in Fig. 4. We only show the edges that have positive labels and are incident on both an A and a B node. In general, there are many edges possible between two nodes; hence it is called a multigraph. For example, we have two directed edges, one labeled by 2 the other by 16, from $B3$ to $A4$.

It is well-known [25] that a Hamiltonian path in the overlap multigraph is a *shortest common superstring* of the sequences in F. Therefore, it is a solution to the genome sequencing problem.

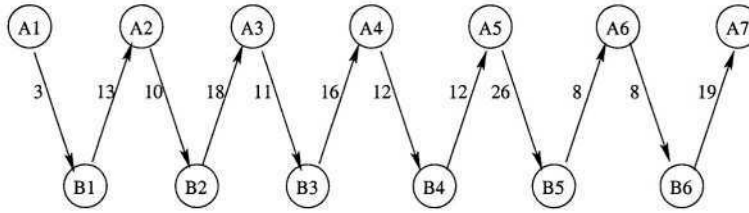

Fig. 5. A Hamiltonian path in the overlap multigraph.

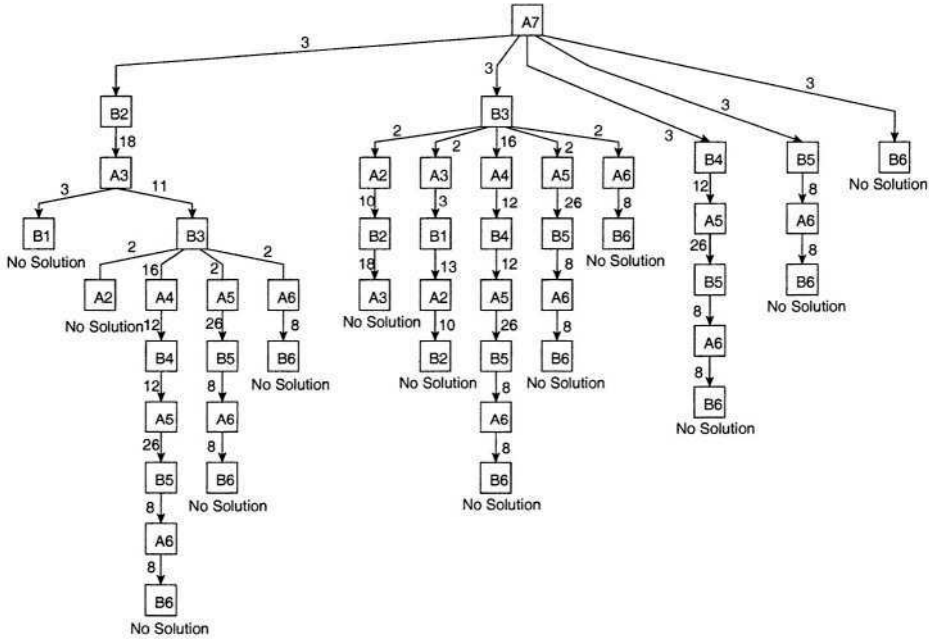

Fig. 6. Greedy search from node $A7$ of the overlap multigraph.

For example, a Hamiltonian path is shown in Fig. 5. This Hamiltonian path matches the original genome sequence in Fig. 1, and incidentally also gives a genome map in terms of As on the top and in terms of Bs on the bottom.

Unfortunately, such a Hamiltonian path is hard to find. A typical greedy search of the graph would always choose to explore the edges with the highest label. The greedy search would start at node $A5$ because it has the highest label, namely 26, on one of its edges. However, $A5$ is a failure node. Then the greedy search would explore $B6$ because it has the edge with the next highest label, namely 19. Continuing this way, the last node that the greedy search will explore is $A1$ or $A7$. Out of these only $A1$ is a success node. Figure 6 shows that $A7$ is a failure node. It is clear that the greedy search on the overlap multigraph is doing a huge amount of work.

Fingerprints. It is computationally hard to compare and match long nucleotide sequences. *Fingerprinting* is a short-cut method that is often much faster. The ideas is to get a "fingerprint" of each subsequence (like those in Figs. 2 and 3) by applying one or more restriction enzymes and then measuring the lengths of the resulting fragments [29], The multiset or *bag* of fragment lengths are usually unique to each subsequence.

For example, by applying the restriction enzymes `BamHI` (`G^GATCC`) and `AvaI` (`C^YCGRG`), where `Y` means either `C` or *T* and `R` means either *A* or `G`, on the subsequences in Fig. 2, we get the fragments and the bag of their lengths shown in Fig. 7.

A1: CAT	{3}
A2: CGATC TCGGGAGG GATCCATTAT	{5,8,10}
A3: CGATTC CCGGGC TCGGGG GATCCTTCCAT	{6,6,6,11}
A4: CGATGGGC CCGAGGCG GATCCCTACTAT	{8,8,12}
A5: CGATC CCGGGGG GATCCTTAATTC TCGAGAAGGCCTAT	{5,7,12,14}
A6: CGATCAAG GATCCTAT	{8,8}
A7: CGATC CCGAGTC CCGGGAT	{5,7,7}

Fig. 7. Fingerprints of the *A* subsequences.

Revesz' Fingerprint Data. Revesz [20] proposed the following procedure using three restriction enzymes a, b, and c to obtain fingerprint data:

1. Take a copy of the DNA.
2. Apply restriction enzyme a to the copy.
3. Separate the subsequences.
4. For each subsequence apply restriction enzyme $b \cup c$, cutting the subsequence into fragments.
5. Find the lengths of the fragments.
6. Repeat Steps (1–5) using b instead of a and $a \cup c$ instead of $b \cup c$.

Each execution of Steps (1–5) is the generation of a *big-bag.* The subsequences obtained in Step (3) are the bags of the big-bag, and the lenghts of of the fragments obtained in Steps (4-5) are the elements of the bags. Once the DNA sequence is cut into subsequences using the above procedure, all the information about the original order is lost. After analyzing these subsequences, they have to be arranged or assembled into a single set of sequences called a *genome map.*

For example, let a, b and c be the restriction enzymes `ClaI`, `BamHI` and `AvaI`, respectively. Then after applying Step (1) we get the subsequences shown in Fig. 2. After Steps (4-5) we get the bags shown in Fig. 7.

In Step (6) when we repeat Step (2) using b instead of a we get the subsequences shown in Fig. 3. When we also repeat Steps (4-5) by applying the restriction enzymes `ClaI` $\cup$ `AvaI` on the latter set of subsequences, we get the fragments and the bag of their lengths shown in Fig. 8.

B1: CAT CGATC TCGGGAGG	{3,5,8}
B2: GATCCATTAT CGATTC CCGGGC TCGGGG	{10,6,6,6}
B3: GATCCTTCCAT CGATGGGC CCGAGGCG	{11,8,8}
B4: GATCCCTACTAT CGATC CCGGGGG	{12,5,7}
B5: GATCCTTAATTC TCGAGAAGGCCTAT CGATCAAG	{12,14,8}
B6: GATCCTAT CGATC CCGAGTC CCGGGAT	{8,5,7,7}

Fig. 8. Fingerprints of the B subsequences.

Note: In this method some care needs to be taken in the choice of the triplet of restriction enzymes so that they can be applied in any order, that is, they are *compatible.* We explain compatibility more precisely and give some examples in Section 4.1.

Why is this fingerprint data useful? Note that regardless of whether we start with the set of A or the set of B subsequences, after the triplet enzyme cuts are finished we get exactly the same set of fragments. These fragments are just grouped differently by the As and the Bs. Those As and Bs that that have a significant fingerprint overlap will tend to have also a nucleotide sequence overlap. Hence we can align or match the fragments in the As with the fragments in the Bs to find a solution.

For example, Fig. 9 shows an alignment that corresponds to the Hamiltonian path in Fig 5. In Fig. 9 each undirected edge is labeled with a set of values such that there are fragments with those set of lengths in the fingerprints of both incident nodes. The alignment requires that for each node the multiset union of the labels on the edges incident on it is the same as its fingerprint. For example, the multiset union of the labels {6,6,6} and {11} on the edges incident on $A3$ gives the fingerprint {6, 6, 6, 11} of $A3$.

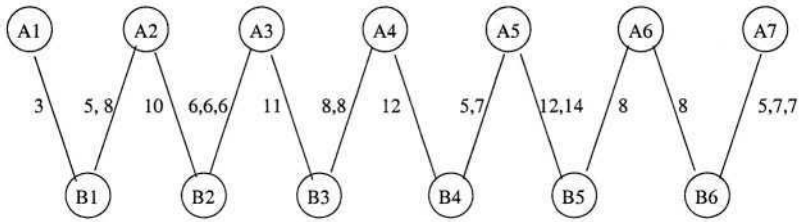

Fig. 9. A fingerprint overlap solution corresponding to the Hamiltonian path in Fig. 5.

We will see in Section 3 a way to find such fingerprint alignments using a constraint automaton.

Constraint Automata. *Constraint automata* are used to control the operation of systems based on conditions that are described using constraints on variables (see Chapter 6 of [21]). A constraint automaton consists of a set of states, a set of state variables, transitions between states, an initial state, and the domain

and initial values of the state variables. Each transition consists of a set of constraints, called the guard constraints, followed by a set of assignment statements. The guard constraints are followed by question marks "?", and the assignment statements are shown using the symbol ":=".

A constraint automaton can move from one state to another if there is a transition whose guard constraints are satisfied by the current values of the state variables. The transitions of a constraint automaton may contain variables in addition to state variables. These variables are said to be existentially quantified variables, which means that some values for these variables must be found such that the guard constraints are satisfied and the transition can be applied.

GMAP Using Constraint Automata. The Genome Map Assembly Problem can be solved using a constraint automaton [20]. The constraint automaton uses an abstraction of the GMAP called a *Big-Bag Matching Problem,* which is NP-complete [23].

This paper implements the Genome Map Assembly automaton and empirically investigates whether the time complexity for the average case is linear or exponential.

2.1 Related Work

The overlap multigraph method has several variations depending on how the subsequences are generated. Random substrings are picked from the DNA in Gillett et al. [6], Olsen et al. [18], Revesz [22], and Wong et al. [31] (which use restriction enzymes) or in Green and Green [7] and Harley et al. [9] (which use a technique called hybridization to test whether two substrings overlap). Lander and Waterman [16] gives some estimates about the number of random substrings of certain average size required to get a good coverage for a DNA of a given size. Unfortunately, the random-substrings approach has the limitation that there could be some gaps on the DNA, i.e., regions that are not covered by any randomly picked substring.

Derrida and Fink [3], Venter et al. [28], and Weber and Myers [30] developed the *whole-genome shotgun sequencing method,* which is another version of the overlap multigraph method applied to the entire genome.

Veeramachaneni et al. [27] abstract the problem of aligning fragmented sequences as an optimization problem and show the problem to be MAX-SNP hard. They also develop a polynomial time algorithm for their method. Kahveci and Singh [11] consider the problem of interactive substring searching using two different techniques: *local statistics-based interactivity* (LIS) and *global statistics-based interactivity* (GIS). These techniques have only 75 % accuracy [12].

One other abstraction of the genome map assembly problem, called the probed partial digestion problem, is considered in Tsur et al. [26]. Partial digestion occurs when a restriction enzyme fails to cut all the sites specific to it. Pop et al. [19] is a survey on the various genome sequence assembly algorithms.

Karp [14] shows several variations of the overlap multigraph problem to be NP-complete. Revesz [23] shows that the *Big-Bag Matching Problem* abstraction of the GMAP is also NP-complete.

3 Constraint Automata Solution

We now describe and illustrate the constraint automata solution of Revesz [20]. The original solution was a pure fingerprints-based method. The modification of *disambiguation* by checking the actual nucleotide strings in case of ambiguity in the fingerprint data during a search is a new idea we introduce in this paper.

Section 3.1 describes the constraint automata. The original automata was described as a non-deterministic automaton, but we give a presentation that is deterministic and uses the concept of "backtracking." Section 3.2 describes the implementation of the constraint automaton in the Perl programming language.

3.1 The Constraint Automata for GMAP

The working of the constraint automaton can be described easiest, if we first abstract the GMAP as a *Big-Bag Matching Problem.*

A *bag* is a multiset, a generalization of a set in which each element can occur multiple times [20]. A *big-bag* is a multiset whose elements are bags that can occur multiple times [20]. For example all the *A*s can be put into a bag. Then when we replace each A_i by the bag describing its fingerprint, then we obtain a big-bag. Similarly, we can obtain a big-bag corresponding to the set of *B*s.

Each permutation of the bags and permutation of the elements of each bag within a big-bag is called a *presentation* [20]. A big-bag can have several different presentations. The *big-bag matching decision problem* (BBMD) is the problem of deciding whether two big-bags match [20]. The *big-bag matching problem* (BBM) is the problem of finding matching presentations for two given big-bags if they match [20].

The following constraint automaton, shown in Fig. 10, uses "backtracking" and can give all the possible solutions.

The automaton uses the constraints "$\subseteq$" (subset) and "–" (set difference). It starts in the INIT state and ends in the HALT state. From the INIT state, the automaton moves from left to right by adding either bag A or B. Each bag represents a *contiguity constraint* for the elements it contains, i.e., any valid presentation must contain the elements within a bag next to each other. Therefore, adding a new bag really adds a new contiguity constraint to a set of other such constraints. Only if the set of constraints is solvable (and there could be several solutions) is the automaton allowed to continue with the next transition.

At any point either the list of A bags will be ahead of the list of B bags ("A_ahead" state) or vice versa ("B_ahead" state) or neither will be ahead, in which case it goes to the INIT state. The automaton has the following states: INIT, A_ahead, B_ahead, HALT and Backtrack. The state variables are: *UA* and *UB* indicating the set of unused *A* and *B* bags, respectively, *S* indicating the set of elements by which either the *A* or the *B* list is currently ahead, *Choices* indicating the set of options from which the next bag can be chosen, *SelBag* indicating the bag that was selected from *Choices* as the next bag, and *Cflag* indicating whether *Choices* is empty or not (if empty then it is set to "0", else it is set to "1"). The value of each state variable is saved after each transition.

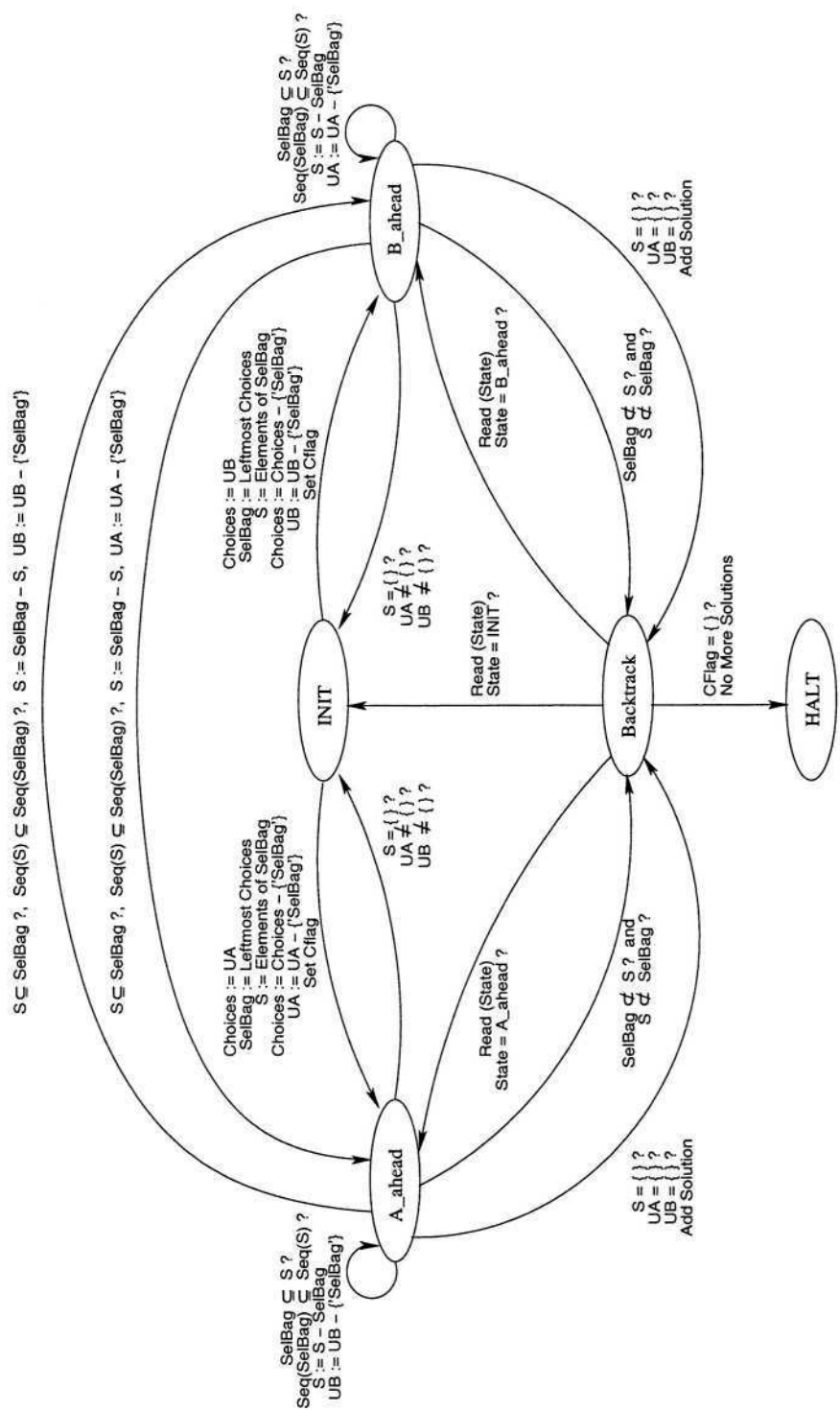

Fig. 10. Constraint Automata Solution.

The automaton can be executed in ONE mode to find the first solution or ALL mode to find all the possible solutions. The working of the automaton is explained below.

1. The automaton is in the "INIT" state. The next bag is chosen from the list that has the lesser number of bags. If the next bag to be chosen is from the list of *A* bags then *Choices* will contain *UA,* else it will contain *UB.* The leftmost bag of *Choices* is removed from *Choices* and set to *SelBag,* the *Cflag* is set to "0" if *Choices* is empty or "1" if it is not empty, the elements of *SelBag* is set to *S,* and the automaton moves to either the "A_ahead" or the "B_ahead" state.
2. The set of options from which the next bag can be chosen is found by determining the bags which are either a subset or a superset of *S. Choices* contains this set of options.
3. If the automaton is in the "A_ahead" state, then the set of options is determined by *UB. SelBag* is the leftmost bag in *Choices.* We now need a *disambiguation* step to cut down on the choices. The nucleotide sequence corresponding to each element in *S* is compared with the nucleotide sequence of the corresponding element in *SelBag.* If they do not match, then the automaton moves to the "Backtrack" state. If the sequences of all the elements match, then
   (a) if $SelBag \subseteq S$, then the automaton remains in the "A_ahead" state and $S := S - SelBag$.
   (b) if $S \subseteq SelBag$, then the automaton moves to the "B_ahead" state and $S := SelBag - S$.

   In both cases, the elements of *S* and *SelBag* are matched as far as possible.
4. If the automaton is in the "B_ahead" state, then the set of options is determined from *UA* and the process followed is similar to that in Step 3.
5. If, in Steps 3 and 4, the difference of the values of *SelBag* and *S* is an empty set and *UA* and *UB* are not empty, then the automaton moves to the "INIT" state. If the difference is an empty set and *UA* and *UB* are also empty, then a solution has been found for the problem. If the execution of the automaton is in ONE mode then it moves to the "HALT" state and stops. If it is executed in ALL mode then the solution is saved and the automaton moves to the "Backtrack" state.
6. If, in Steps 3 and 4, *SelBag* is neither a subset nor a superset of *S,* then the automaton moves to the "Backtrack" state. In this state, the automaton will check for the last node whose *Cflag* has the value "1". The automaton then backtracks to this node and the information saved at this node is retrieved. Based on this information, the automaton will move to one of the states. If none of the nodes have their corresponding *Cflag* set to "1", then there is no more solution possible for the problem and the automaton moves to the "HALT" state and stops.

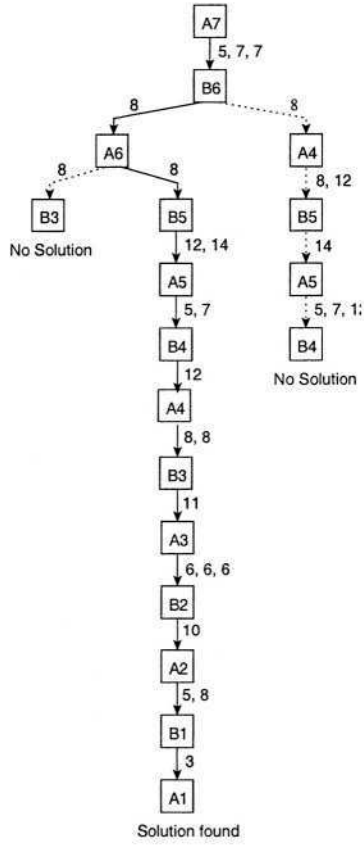

Fig. 11. The constraint automaton search tree from node *A*7.

3.2 Implementation

The working of the automaton is similar to a pre-order traversal of a tree. Every node in the tree is a bag of either of the two big-bags. The two big-bags shown in Figs. 7 and 8 are used to explain the working of the automaton. The corresponding search tree is shown in Fig. 11. The dashed lines are part of the search tree only if we skip the disambiguation test. In the following we assume that we skip disambiguation. A partial pre-order traversal of the tree is shown step-wise in Table 1.

The execution starts at the origin (Node 0), which is the initial state (INIT). The set *S* is empty. The various options from which the automaton can choose a starting bag are all the bags in *UA*. Hence, the set *Options* contain all the bags of *UA*. A7 is taken as the starting bag *(SelBag)* and the remaining bags (*A*1...*A*6) are put into *Choices.* Since *Choices* is not empty, the flag *Cflag* for that node is set to "1". This means that there is another branch possible from this node.

Table 1. Step-wise partial pre-order traversal of the tree in Fig. 11.

Node	CurrBag	S	UA	UB	State	Options	SelBag	Choices	Cflag
0	-	{}	A1,A2,A3,A4,A5,A6,A7	B1,B2,B3,B4,B5,B6	INIT	A1,...,A7	A7	A1,...,A6	1
1	A7	{5,7,7}	A1,A2,A3,A4,A5,A6	B1,B2,B3,B4,B5,B6	A_ahead	B6	B6	{}	0
2	B6	{8}	A1,A2,A3,A4,A5,A6	B1,B2,B3,B4,B5	B_ahead	A4,A6	A4	{A6}	1
3	A4	{8,12}	A1,A2,A3,A5,A6	B1,B2,B3,B4,B5	A_ahead	B5	B5	{}	0
4	B5	{14}	A1,A2,A3,A5,A6	B1,B2,B3,B4	B_ahead	A5	A5	{}	0
5	A5	{5,7,12}	A1,A2,A3,A6	B1,B2,B3,B4	A_ahead	B4	B4	{}	0
6	B4	{}	A1,A2,A3,A6	B1,B2,B3,B4	Backtrack	-	-	-	-
Cflag != 0; Backtrack to Node 2									
2	B6	{8}	A1,A2,A3,A4,A5,A6	B1,B2,B3,B4,B5	B_ahead	A6	A6	{}	0
3	A6	{8}	A1,A2,A3,A4,A5	B1,B2,B3,B4,B5	A_ahead	B3,B5	B3	{B5}	1
4	B3	{8,11}	A1,A2,A3,A4,A5	B1,B2,B4,B5	Backtrack	-	-	-	-
Cflag != 0; Backtrack to Node 3									
3	A6	{8}	A1,A2,A3,A4,A5	B1,B2,B3,B4,B5	A_ahead	B5	B5	{}	0
4	B5	{12,14}	A1,A2,A3,A4,A5	B1,B2,B3,B4	B_ahead	A5	A5	{}	0
5	A5	{5,7}	A1,A2,A3,A4	B1,B2,B3,B4	A_ahead	B4	B4	{}	0
6	B4	{12}	A1,A2,A3,A4	B1,B2,B3	B_ahead	A4	A4	{}	0
7	A4	{8,8}	A1,A2,A3	B1,B2,B3	A_ahead	B3	B3	{}	0
8	B3	{11}	A1,A2,A3	B1,B2	B_ahead	A3	A3	{}	0
9	A3	{6,6,6}	A1,A2	B1,B2	A_ahead	B2	B2	{}	0
10	B2	{10}	A1,A2	B1	B_ahead	A2	A2	{}	0
11	A2	{5,8}	A1	B1	A_ahead	B1	B1	{}	0
12	B1	{3}	A1	{}	B_ahead	A1	A1	{}	0
13	A1	{}	{}	{}	Backtrack	-	-	-	-
Cflag != 0; Backtrack to Node 1									

The selected bag, *A*7, is then the node 1 of the tree and the current bag (*CurrBag*). *S* now contains the elements of bag *A*7. *UB* contains all the bags of big-bag *B* and *UA* contains all the bags of big-bag *A* except *A*7, which has been used. Since *S* contains the elements by which big-bag *A* is ahead of big-bag *B,* the automaton is in the "A_ahead" state. The next bag to be chosen is from *UB,* which contains the unused *B* bags. There is only one possible choice: *B*6. So we select bag *B*6. *Choices* is now empty, so we set *Cflag* to "0". Since the elements of *S* are contained in *B*6, we subtract the elements of *S* from the elements of *B*6. The resulting *S* will contain the elements by which big-bag *B* is ahead. So the automaton will move to the "B_ahead" state. Bag *B*6 is then removed from *UB*.

The execution continues in a similar fashion until it reaches node 6. At this node *S* is empty and there are no possible solutions. So the automaton moves to the "Backtrack" state. From the table we can see that the last node to have its *CFlag* set to "1" is node 2. So the automaton backtracks to that node. The bag in *Choices* is set to *SelBag.* Now *Choices* is empty, so *Cflag* is set to "0". The execution then continues as explained above.

At node 13, *S, UA* and *UB* are empty. Hence a solution is found. If the automaton is executed in ONE mode then it moves into the "HALT" state and the execution stops. If the execution is in ALL mode, then the solution is saved and the automaton moves into the "Backtrack" state. The last node to have its *Cflag* set to "1" is found and the automaton backtracks to that node.

Note that the search tree in Fig. 11 is much smaller than the search tree in Fig. 6, although both start from the same node *A*7. The search tree of Fig. 11 is even smaller, with the dashed lines deleted, if we use the disambiguation test. In general the constraint automaton will be smaller than the greedy search tree in the overlap multigraph for each starting node with a few exceptions.

There is also an interesting difference in the outcomes of the search when starting from node *A*7. The overlap multigraph method fails as shown in Fig 6. However, the constraint automaton search succeeds and returns the following sequence of A nodes:

$$A7\ A6\ A5\ A4\ A3\ A2\ A1$$

which is exactly the reverse order that was input. Like in nature, the information about left-to-right or right-to-left ordering is lost after restriction enzyme cuts. In general, the constraint automaton search can succeed starting from two nodes instead of only one in the overlap multigraph method. That also helps to increases the relative speed of the constraint automaton.

4 Results and Analysis

All experiments were carried out on an Athlon XP 1800+ desktop with 512 MB RAM and running the Windows XP Professional operating system. Perl v5.8.0 was used to implement the generation of input and the constraint automaton.

The programs were executed in *Cygwin* v1.5.5-1, a Linux-like environment for Windows.

The input generation Perl program is executed on 70 sequences of the first four chromosomes of the mouse genome along with five sets of non-overlapping restriction enzyme triples [24], to generate the input big-bags. The lengths of the 70 sequences varied from 1.2 million to 36.8 million base pairs. The Perl implementation of the constraint automata takes these inputs and can be set by the user to find either only one or all possible solutions. The running times were measured for finding one solution.

4.1 Generation of Data and Results

The DNA sequences for generating the input for the Constraint Automata were taken from the Mouse Genome Resources website of the National Center for Biotechnology Information (NCBI) [17]. These sequences are various parts of the chromosomes of the mouse DNA. These sequences are subjected to restriction enzymes that are *compatible* with each other, that is, the recognition sequences of the enzymes do not overlap. For example, the three restriction enzymes `ClaI` (`AT^CGAT`), `BamHI` (`G^GATCC`), and `AvaI` (`C^YCGRG`) are compatible.

An example of a *non-compatible* restriction enzyme triple is `EcoRV` (`GAT^ ATC`), `ClaI` (`AT^CGAT`) and `BamHI` (`G^GATCC`). That is non-compatible because the recognition sequence of `EcoRV` overlaps with the recognition sequence of `ClaI`. One overlap is that `ATC` is both the ending sequence of `EcoRV` and the starting sequence of `ClaI`. Another overlap is that `GAT` is both the ending of sequence of `ClaI` and the starting sequence of `EcoRV`.

If we use more than three restriction enzymes, there will be a large number of fragments. On the other hand, if we use less than three restriction enzymes, the number of fragments will be too small, resulting in huge fragment lengths. So we use three restriction enzymes to obtain a normal number of fragments, with a normal length.

The 70 sequences are subjected to the following five sets of non-overlapping restriction enzyme triples, which were carefully selected from a restriction enzyme database [24], to give us the input big-bags for the constraint automaton.

1. `HindIII`, `AccI`, `AceI`
2. `AauI`, `HindII`, `CacBI`
3. `BclI`, `AhyI`, `TaaI`
4. `PsiI`, `PciI`, `HhaII`
5. `PdiI`, `SurI`, `SspI`

The Perl implementation of the constraint automaton is then run using the input bags generated for each of the 70 sequences and the CPU time taken to find the first solution is calculated. This time is noted as the execution time. Further, the series of execution times is used as an indication of the time complexity of the constraint automata.

4.2 Data Analysis and Charts

Execution time data was subject to preliminary data cleaning procedures, which included checking for outliers and normality of distribution (i.e., skew) as per Hoaglin et al. [10] recommendations. Windsorizing procedures [10] were used for outlier analysis. Thus "too extreme" values were replaced with the "most extreme acceptable value". This has the advantage of not losing any data.

To investigate if the time complexity of the constraint automata solution is linear or exponential, we examined the relationship between the input for the constraint automata solution (number of bags) and the execution time. Using curve estimation regression modeling in SPSS, execution time was regressed onto the number of bags for each restriction enzyme triple. Thus five regression models were constructed, one for each restriction enzyme triple. Linear and exponential functions were plotted for each regression model and goodness of fit was determined by variance (R^2) estimates (Table 2) and by visual inspection of the charts (Figs. 12, 14, 16, 18, 20).

Table 2. R^2 values for the regression of execution time on number of bags.

Rest. Enz.	Constraint Automata Solution			Overlap Multigraph		
Triple	Linear	Quadratic	Exponential	Linear	Quadratic	Exponential
1	0.897	0.872	0.814	0.869	0.931	0.959
2	0.908	0.869	0.804	0.875	0.860	0.936
3	0.976	0.846	0.825	0.904	0.868	0.942
4	0.963	0.902	0.865	0.874	0.925	0.956
5	0.909	0.848	0.819	0.887	0.877	0.937
Avg R^2	0.925	0.867	0.825	0.882	0.892	0.946

As seen in Table 2, a linear function provides a better fit (higher variance or R^2) than either a quadratic or an exponential function for all the restriction enzyme triples. A visual inspection of the charts in Figs. 12, 14, 16, 18, 20 also shows that the linear function is the best fit.

The corresponding model for the overlap multigraph is shown in the charts in Figs. 13, 15, 17, 19, 21. From these charts we see that an exponential function provides the best fit for all the restriction enzyme triples.

5 Relation to Constraint Databases

It is well-known that a constraint automaton can be translated into a Datalog query on a constraint database [13,15,21] that contains the input data. In our case the Datalog query needs only Boolean algebra equality, inequality and precedence constraints, where the particular Boolean algebra used is the one whose elements are finite subsets of the natural numbers and whose operators

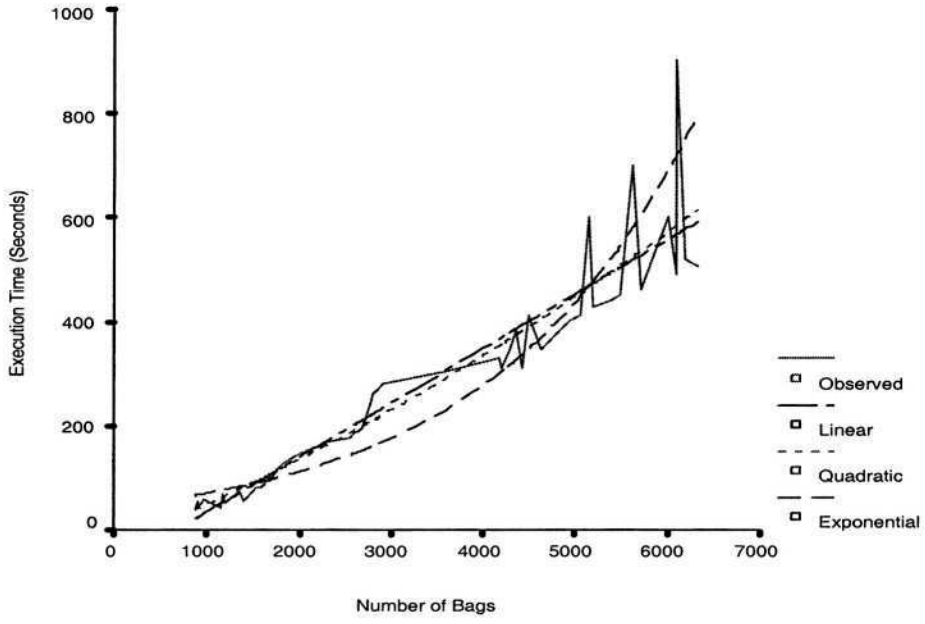

Fig. 12. Regression model for RE triple 1 for constraint automata.

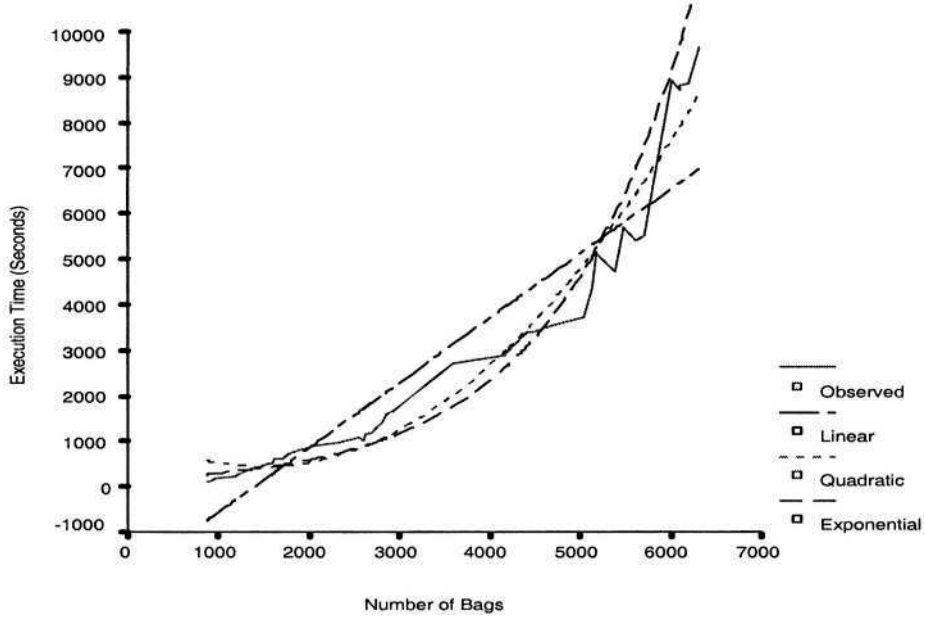

Fig. 13. Regression model for RE triple 1 for overlap multigraph.

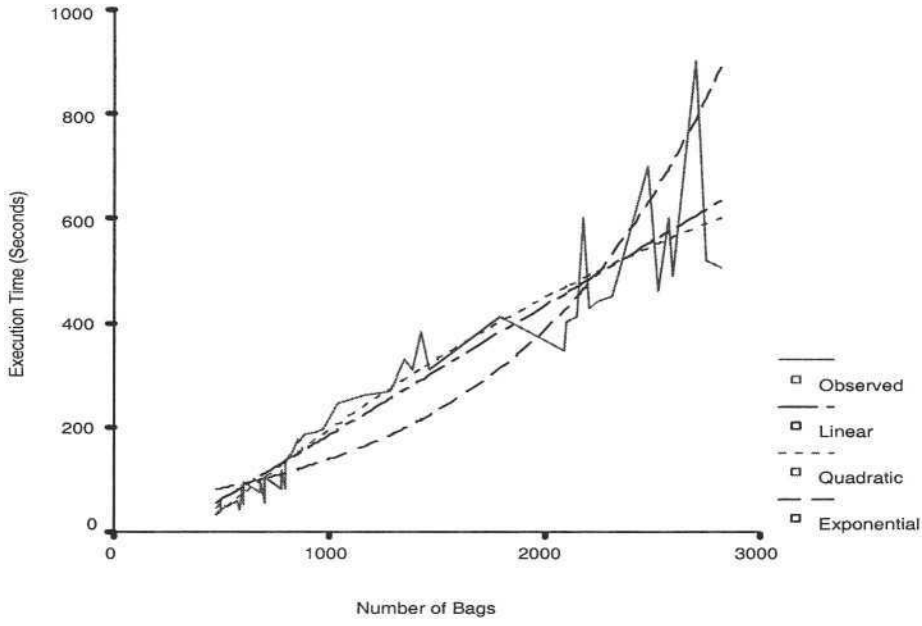

Fig. 14. Regression model for RE triple 2 for constraint automata.

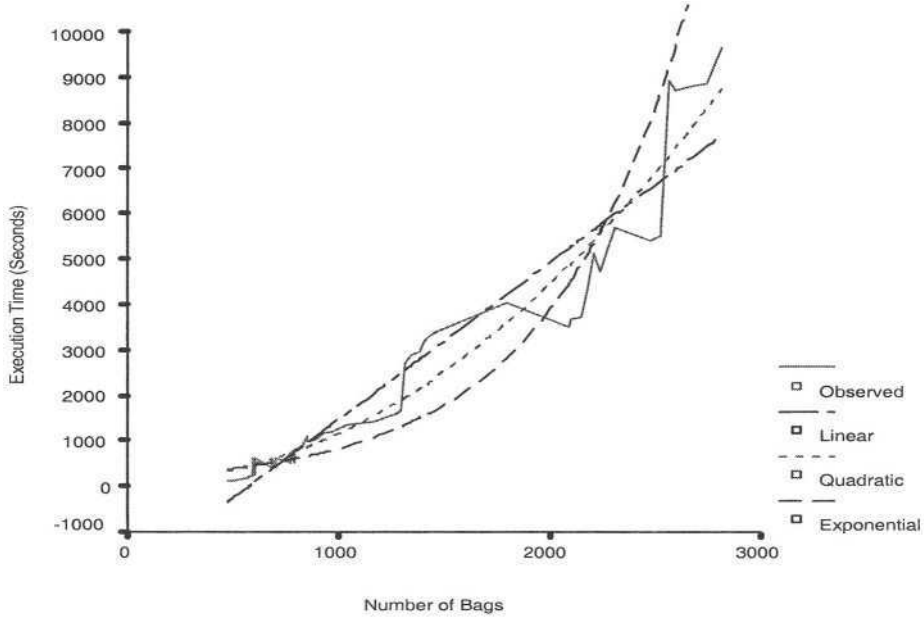

Fig. 15. Regression model for RE triple 2 for overlap multigraph.

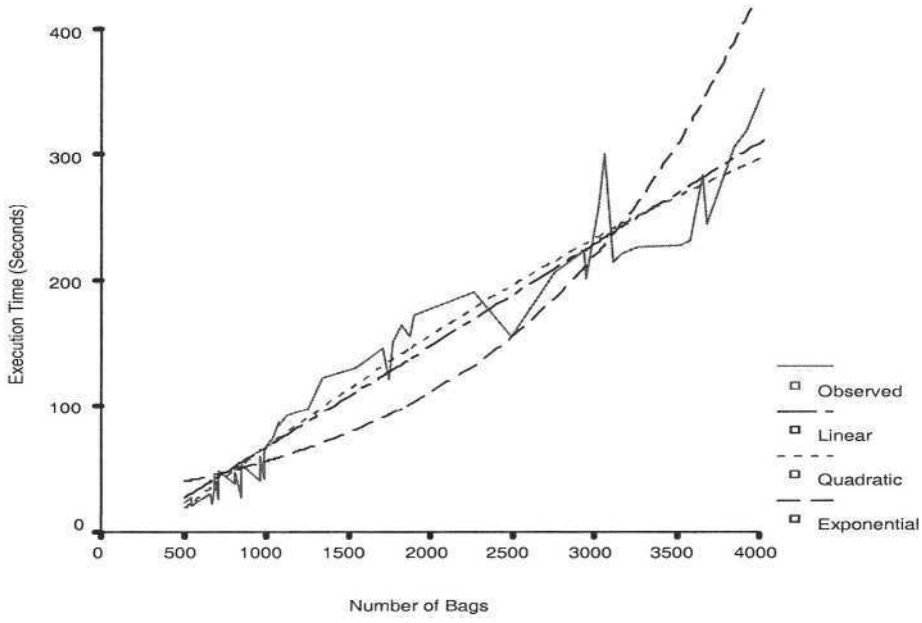

Fig. 16. Regression model for RE triple 3 for constraint automata.

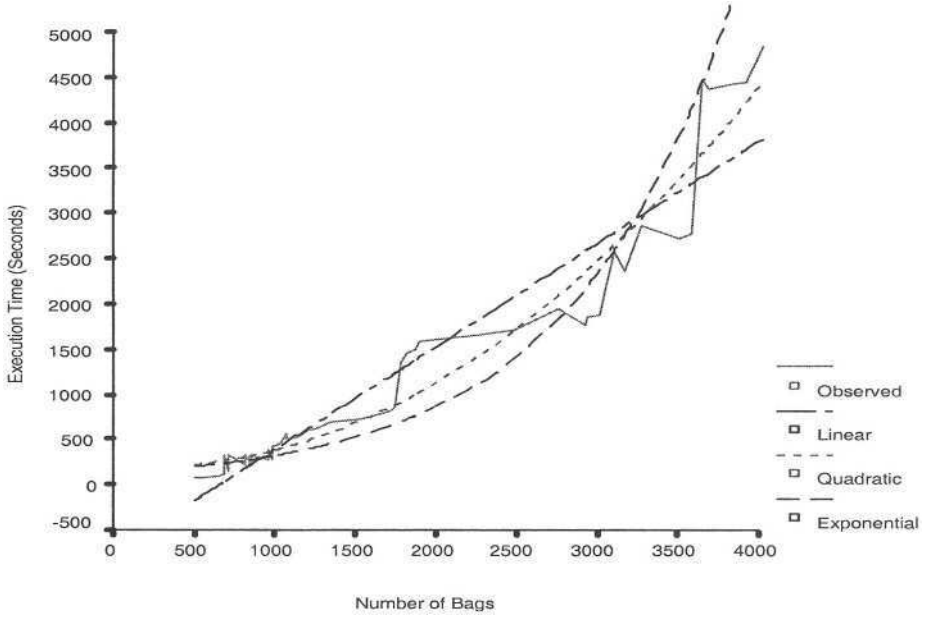

Fig. 17. Regression model for RE triple 3 for overlap multigraph.

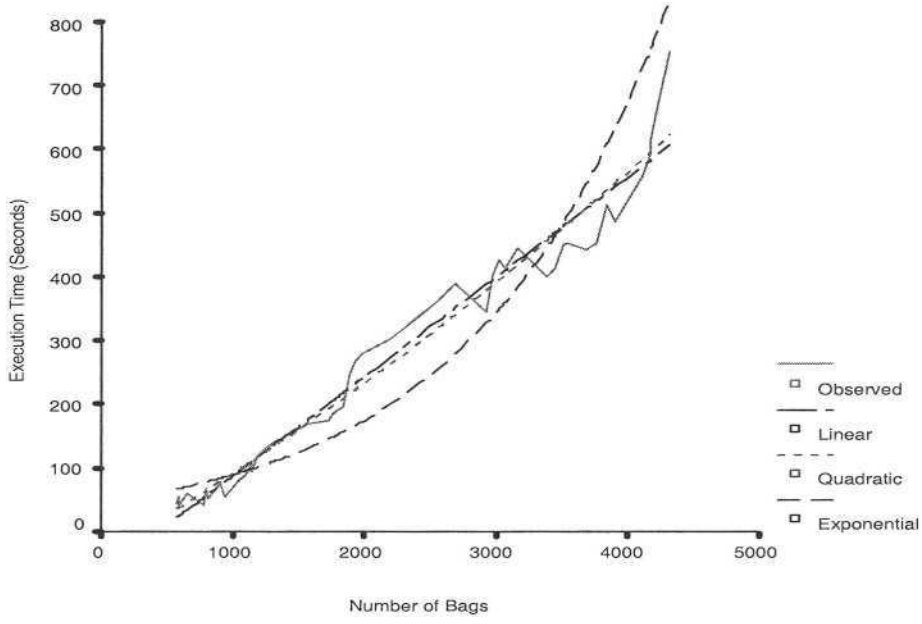

Fig. 18. Regression model for RE triple 4 for constraint automata.

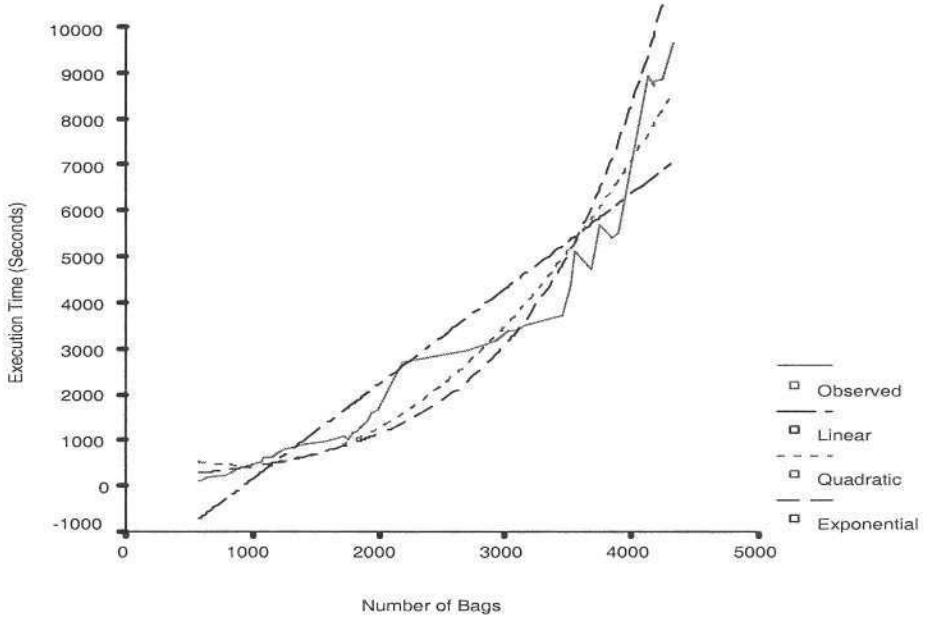

Fig. 19. Regression model for RE triple 4 for overlap multigraph.

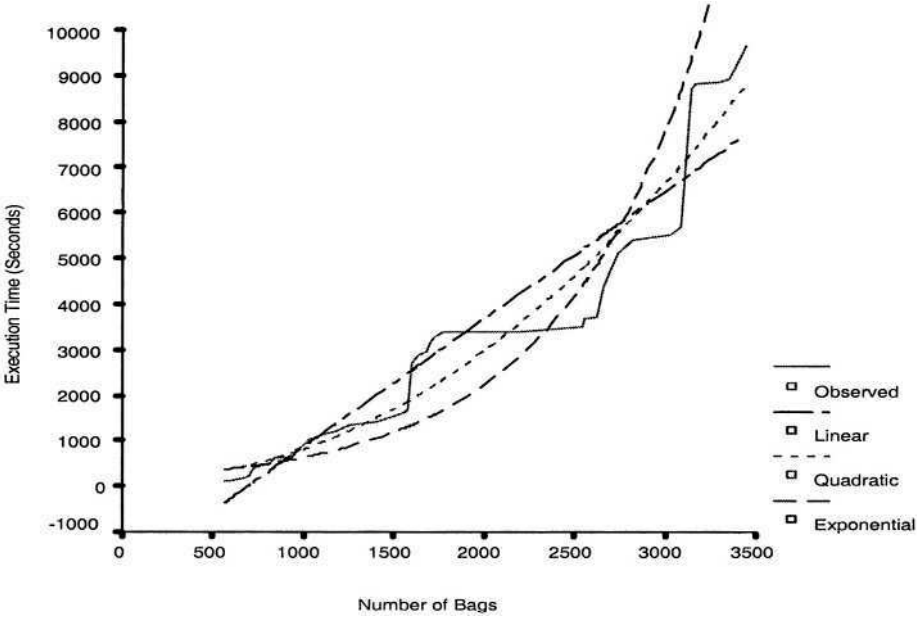

Fig. 20. Regression model for RE triple 5 for constraint automata.

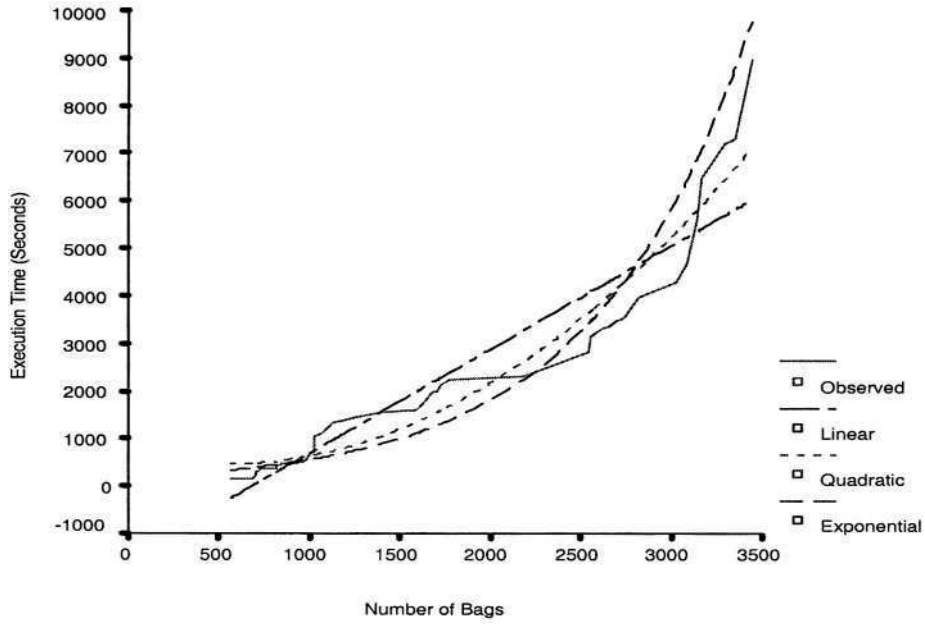

Fig. 21. Regression model for RE triple 5 for overlap multigraph.

are interpreted as the set union, intersection and complement with respect to the set of natural numbers.

The latest version of the DISCO constraint database system [2] implements constraints over the Boolean algebra of the powerset of natural numbers. Hence we could have also implemented the constraint automaton in the DISCO system, although the solution would have been slower, and we could not have modified it as we liked to control backtracking. We chose Perl for the implementation of the constraint automaton, because it allowed us to control backtracking, including the occasional look-up of the nucleotide sequence for disambiguation, and because it is currently a standard language for genomics implementations.

Although our choice may disappoint some constraint database purists, they may take heart in the fact that constraint databases were useful in the discovery of the algorithm. It is by attempting to describe the GMAP in a Boolean algebra of sets, that the abstraction of the entire problem as a Big-Bag Matching Problem was discovered. After that the constraint automaton solution was a easy and natural step. Further, early prototyping in the DISCO system convinced us that the algorithm may be practical in practice and led to the present paper. Hence constraint databases played an essential part in the discovery of the algorithm.

Our experience illustrates a general principle that fewer tools are often better. A general purpose programming language provides too many tools for a programmer to work with. That is not always as helpful as it may seem at first glance, because one is lost in the enormous number of options, the endless number of possible data structures to implement and the also endless number of possible functions to implement on them. With a constraint language, one is not so lost. The programmer has the tool of the constraint language itself and is hence forced to express the problem using a high-level constraint abstraction.

This process either works or does not. As the old saying goes, if you have a hammer every problem starts to look like a nail. If the problem is indeed like a nail, then everything is fine. If it is not, then a new tool, in this case a new programming language, is needed. But that does not mean that starting with a hammer was a bad thing. For without the hammer we may not start to see even nail-like problems as such. The GMAP is a problem that can be expressed using Boolean algebra constraints and a simple automaton. This is a useful abstraction that one who only works with Perl may not discover so easily. Hence the process worked in the case of GMAP.

6 Conclusions and Future Work

We examined the relationship between the input for the constraint automata solution (number of bags) and the execution time using curve estimation regression modeling. This was done for five compatible restriction enzyme triples. Both variance estimates (R^2) and a visual inspection of the charts indicate that a linear function provides the best fit for the constraint automaton and an exponential provides the best fit for the overlap multigraph solution for the GMAP. The experimental results provide strong support for the earlier hypothesis that

the constraint automaton solution is better than the overlap multigraph solution for large genomes.

In the future, we still need to evaluate the applicability of the constraint automaton solution to genome mapping in real biochemical laboratories. The constraint automaton can be further adapted to incorporate *error tolerance* as the actual DNA fragments and their length measures are not necessarily error-free. For example, if a fingerprint is {97,120,355} and another fingerprint is {96,121,357}, then the two can be matched with an error tolerance of 2. The time complexity function for the error-prone data can then be compared with the time complexity for the error-free data. As the error tolerance value increases, the choices in the search increase slowing down the execution time of the constraint automaton. However, it is hypothesized that the constraint automaton with error tolerance will also show a linear time complexity on average.

We also discussed the advantages of trying to attack a problem using multiple programming languages. The more different are the programming languages, the greater is the potential benefit in trying each of them. The constraint automaton solution was discovered with ease after trying Datalog with Boolean algebra constraints. We hope that others will also experience with delight the ease of solving problems in constraint databases.

References

1. J.M. Berg, J.L. Tymoczko, and L. Stryer. *Biochemistry, 5 ed.* W.H. Freeman, New York, 2002.
2. J. Byon and P. Revesz. DISCO: A Constraint Database with Sets. *Proc. Workshop on Constraint Databases and Applications,* Springer LNCS 1034, pp. 68-83, 1995.
3. B. Derrida and T.M.A. Fink. Sequence determination from overlapping fragments: A simple model of whole-genome shotgun sequencing. In *Physical Review Letters,* 88(6):068106, 2003.
4. S.E. DeWeerdt. *What's a Genome?* The Center for Advanced Genomics, 2003.
5. Entrez Database. http://www.ncbi.nlm.nih.gov/entrez/
6. W. Gillett, L. Hanks, G.K-S. Wong, J. Yu, R. Lim, and M.V. Olsen. Assembly of high-resolution restriction maps based on multiple complete digests of a redundant set of overlapping clones. *Genomics,* 33:389-408, 1996.
7. E.D. Green and P. Green. Sequence-tagged site (STS) content mapping of human chromosomes: Theoretical considerations and early experiences. *PCR Methods and Applications,* 1:77-90, 1991.
8. E. Harley and A.J. Bonner. A flexible approach to genome map assembly. In Proc. *International Symposium on Intelligent Systems for Molecular Biology,* pages 161-69. AAAI Press, 1994.
9. E. Harley, A.J. Bonner, and N. Goodman. Good maps are straight. In *Proc. 4th International Conference on Intelligent Systems for Molecular Biology,* pages 88-97, 1994.
10. D.C. Hoaglin, F. Mosteller, and J.W. Tukey. *Understanding Robust and Exploratory Data Analysis.* John Wiley, New York, 1983.
11. T. Kahveci and A.K. Singh. Genome on demand: Interactive substring searching. In *Proceedings of the Computational Systems Bioinformatics.* IEEE Computer Society, 2003.

12. T. Kahveci and A.K. Singh. An interactive search technique for string databases. Technical Report 10, UCSB, 2003.
13. P. Kanellakis, G. Kuper, and P. Revesz. Constraint query languages. *Journal of Computer and System Sciences,* 51:26-52, 1995.
14. R.M. Karp. Mapping the genome: Some combinatorial problems arising in molecular biology. In *Proc. 25th ACM Symposium on Theory of Computing,* pages 278-85. ACM Press, 1993.
15. G. Kuper, L. Libkin and J. Paredaens: *Constraint Databases,* Springer, 2000.
16. E.S. Lander and M.S. Waterman. Genomic mapping by fingerprinting random clones: A mathematical analysis. *Genomics,* 2:231-9, 1988.
17. Mouse Genome Resources. http://www.ncbi.nlm.nih.gov/genome/guide/mouse/
18. M.V. Olson, J.E. Dutchik, M.Y. Graham, G.M. Brodeur, C. Helms, M. Frank, M. MacCollin, R. Scheinman, and T. Frank. Random-clone strategy for genomic restriction mapping in yeast. *Genomics,* 83:7826-30, 1986.
19. M. Pop, S.L. Salzberg, and M. Shumway. Genome Sequence Assembly: Algorithms and Issues. *Computer,* pages 47-54, July 2002.
20. P. Revesz. Bioinformatics. In *Introduction to Constraint Databases,* pages 351-60. Springer, New York, 2002.
21. P. Revesz. *Introduction to Constraint Databases,* Springer, New York, 2002.
22. P. Revesz. Refining restriction enzyme genome maps. *Constraints,* 2(3-4):361-75, 1997.
23. P. Revesz. The dominating cycle problem in 2-connected graphs and the matching problem for bag of bags are NP-complete. In *Proc. International Conference on Paul Erdos and His Mathematics,* pages 221-5, 1999.
24. R. J. Roberts. *REBASE: The Restriction Enzyme Database.* New England Biolabs, http://rebase.neb.com/rebase/rebase.html, 2003.
25. J. Setubal and J Meidanis. Fragment Assembly of DNA. In *Introduction to Computational Molecular Biology* pages 118-24, PWS Publishing, Boston, 1997.
26. S. Tsur, F. Olken, and D. Naor. Deductive databases for genome mapping. In *Proc. NACLP Workshop on Deductive Databases,* 1993.
27. V. Veeramachaneni, P. Berman, and W. Miller. Aligning Two Fragmented Sequences. *Discrete Applied Mathematics,* 127(1): 119-43, 2003.
28. J.C. Venter, M.D. Adams, G.G. Sutton, A.R. Kerlavage, H.O. Smith, and M. Hunkapiller. Shotgun sequencing of the human genome. *Science,* 280:1540-2, 1998.
29. D. Voet and J. Voet. *Biochemistry, 3 ed., Vol. 1.* John Wiley, New York, 2003.
30. J.L. Weber and E.W. Myers. Human whole-genome shotgun sequencing. *Genome Research,* 7(5):401-9, 1997.
31. G.K-S. Wong, J. Yu, E.C. Thayer, and M.V. Olson. Multiple-complete-digest restriction fragment mapping: Generating sequence-ready maps for large-scale DNA sequencing. In *Proc. National Academy of Sciences, USA,* 94(10):5225-30, 1997.

Dynamic FP-Tree Based Mining of Frequent Patterns Satisfying Succinct Constraints

Carson Kai-Sang Leung

The University of Manitoba
Winnipeg, MB, Canada
kleung@cs.umanitoba.ca

Abstract. Since its introduction, frequent-pattern mining has been generalized to many forms, which include *constrained data mining*. The use of *constraints* permits user focus and guidance, enables user exploration and control, and leads to effective pruning of the search space and efficient mining of frequent patterns. In this paper, we focus on the use of *succinct constraints*. In particular, we propose a novel algorithm, called *dyFPS*, for *dynamic FP-tree based mining of frequent patterns satisfying succinct constraints*. Here, the term "dynamic" means that, in the middle of the mining process, users are able to modify the succinct constraints they specified. In terms of functionality, our algorithm is capable of handling these modifications effectively by exploiting succinctness properties of the constraints in an FP-tree based framework. In terms of performance, the dyFPS algorithm efficiently computes all frequent patterns satisfying the constraints.

Keywords: Data mining, constraints, succinctness, dynamic changes, FP-trees

1 Introduction

The problem of mining association rules [1,2] – and the more general problem of finding frequent patterns – from large databases has been the subject of numerous studies. Over the past decade, frequent-pattern mining has been generalized to many forms, which include constrained data mining. The use of *constraints* permits users to specify the patterns to be mined according to their intention, and thereby allowing user focus and guidance. Consequently, the computation is limited to what interests the users. In addition, constrained mining also enables user exploration and control. As a result, effective pruning of the search space and efficient mining of frequent patterns can be achieved.

To handle constraints in the process of mining frequent patterns from large databases, many different approaches have been proposed. The following are some examples. Srikant et al. [12] considered item constraints in association rule mining. Bayardo et al. [4] developed Dense-Miner to mine association rules with the user-specified consequent meeting "interestingness" constraints (e.g., minimum support, minimum confidence, minimum improvement). Garofalakis

B. Kuijpers and P. Revesz (Eds.): CDB 2004, LNCS 3074, pp. 112–127, 2004.

Auxiliary information about items:

Item	a	b	c	d	e	f	g
$Price$	36	12	24	28	20	32	16
Qty	700	300	400	500	800	600	200
$Type$	$soda$	$soda$	$snack$	$beer$	$beer$	$beer$	$meat$

$C_1 : min(S.Qty) \geq 500$
$C_2 : max(S.Price) \geq 28$
$C_3 : S.Type \supseteq \{snack, soda\}$
$C_4 : S.Price = 16$
$C_5 : S.Type \subseteq \{beer, snack\}$
$C_6 : soda \in S.Type$
$C_7 : max(S.Price)/avg(S.Price) \leq 7$

Fig. 1. Examples of various classes of constraints

et al. [5] developed SPIRIT to mine frequent sequential patterns satisfying regular expression constraints. Ng et al. [8,10] proposed a constrained frequent-set mining framework within which users can use a rich set of constraints – including SQL-style aggregate constraints (e.g., C_1 and C_2 in Fig. 1) and non-aggregate constraints (e.g., C_3, C_4, C_5 and C_6) – to guide the mining process to find only those rules satisfying the constraints. In Fig. 1, constraint $C_1 \equiv min(S.Qty) \geq 500$ says that the minimum *Qty* value of all items in the set *S* is at least 500. Constraint $C_3 \equiv S.Type \supseteq \{snack, soda\}$ says that the set *S* includes some items whose *Type* is *snack* and some items whose *Type* is *soda;* constraint $C_4 \equiv S.Price = 16$ says that all items in the set *S* are of *Price* equal to 16. Ng et al. also developed the CAP algorithm in the constrained frequent-set mining framework mentioned above. Such an Apriori-based algorithm exploits properties of *anti-monotone constraints* and/or *succinct constraints* to give as much pruning as possible. Constraints such as $C_1, ..., C_6$ in Fig. 1 are **succinct** because one can directly generate precisely all and only those itemsets satisfying the constraints (e.g., by using a precise "formula", called a member generating function, that does not require generation and testing of itemsets not satisfying the constraints). For instance, itemsets satisfying constraint $C_2 \equiv max(S.Price) \geq 28$ can be precisely generated by combining at least one item whose $Price \geq 28$ with some possible optional items (whose *Prices* are unimportant), thereby avoiding the substantial overhead of the generate-and-test paradigm. It is important to note that a *majority of constraints in this constrained frequent-set mining framework is succinct;* for those constraints that are not succinct, many of them can be induced to weaker constraints that are succinct! Among the succinct constraints in Fig. 1, $C_1 \equiv min(S.Qty) \geq 500$ is also **anti-monotone** because any superset of an itemset violating the constraint (i.e., containing an item whose $Qty < 500$) also violates the constraint. Grahne et al. [6] also exploited the anti-monotone and/or succinct constraints, but they mined valid correlated itemsets. Pei et al. [11] developed the $\mathcal{FIC}$ algorithms, which integrate a tree-based mining framework with constraint pushing. Specifically, to enhance performance,

$\mathcal{FIC}$ uses an extended prefix-tree structure – called *Frequent Pattern tree* or *FP-tree* [7] – to capture the content of the transaction database. The algorithms exploit the so-called convertible constraints (e.g., C_7 in Fig. 1). Leung et al. [9] exploited succinct constraints, and developed the FPS algorithm to effectively mine frequent itemsets satisfying succinct constraints. However, their FPS algorithm does not handle dynamic changes to succinct constraints.

It is well-known that data mining is supposed to be a human-centered and exploratory process. Human involvement is not confined to user focus (i.e., constrained mining); it also includes *interactive mining*. With interactive mining, users can (i) monitor the mining process and (ii) make change dynamically during the mining process, and thus having a decisive influence on the mining process.

In this paper, our **key contribution** is the development of a novel algorithm, called *dyFPS,* for *dynamic FP-tree based mining of frequent patterns satisfying succinct constraints.* This algorithm can be considered as a non-trivial extension of the FPS algorithm [9]. In terms of functionality, the dyFPS algorithm allows users to modify the succinct constraints in the middle of the mining process, and it handles these modifications very effectively. In terms of performance, the dyFPS algorithm, like its static counterpart (i.e., the FPS algorithm), is very efficient because it avoids the generate-and-test paradigm by exploiting succinctness properties of the constraints in an FP-tree based framework.

This paper is organized as follows. In the next section, relevant background material is described. Section 3 presents an overview of our proposed dyFPS algorithm. Section 4 shows the experimental results. Finally, conclusions are presented in Section 5.

2 Background

In this section, we first give a definition of succinct constraints. We then provide an overview of the FPS algorithm [9], which is an FP-tree based mining algorithm for handling succinct constraints.

2.1 Succinct Constraints

Definition 1 (Succinctness [10]). *Define* $\text{SAT}_C(\texttt{Item})$ *to be the set of itemsets that satisfy the constraint C. With respect to the lattice space consisting of all itemsets,* $\text{SAT}_C(\texttt{Item})$ *represents the pruned space consisting of those itemsets satisfying C. We use the notation* 2^I *to mean the powerset of I.*

(a) $I \subseteq \texttt{Item}$ *is a* succinct set *if it can be expressed as* $\sigma_p(\texttt{Item})$ *for some selection predicate* p*, where* σ *is the selection operator.*
(b) $SP \subseteq 2^{\texttt{Item}}$ *is a* succinct powerset *if there is a fixed number of succinct sets* $\texttt{Item}_1, ..., \texttt{Item}_k \subseteq \texttt{Item}$ *such that SP can be expressed in terms of the powersets of* $\texttt{Item}_1, ..., \texttt{Item}_k$ *using union and minus.*
(c) A constraint C is **succinct** *provided that* $\text{SAT}_C(\texttt{Item})$ *is a succinct powerset.*

Consider constraint $C_3 \equiv S.Type \supseteq \{snack, soda\}$. Its pruned space consists of all those itemsets that contain at least one *snack* item and at least one *soda* item. Let $\mathtt{Item}_1$, $\mathtt{Item}_2$ and $\mathtt{Item}_3$, respectively, be the sets $\sigma_{Type=snack}(\mathtt{Item})$, $\sigma_{Type=soda}(\mathtt{Item})$ and $\sigma_{Type \neq snack \wedge Type \neq soda}(\mathtt{Item})$. Then, $\mathtt{Item}_1$ contains all the *snack* items, $\mathtt{Item}_2$ contains all the *soda* items, and $\mathtt{Item}_3$ contains neither a *snack* item nor a *soda* item. Hence, C_3 is succinct because its pruned space $\mathrm{SAT}_{C_3}(\mathtt{Item})$ can be expressed as $2^{\mathtt{Item}} - 2^{\mathtt{Item}_1} - 2^{\mathtt{Item}_2} - 2^{\mathtt{Item}_1 \cup \mathtt{Item}_3} - 2^{\mathtt{Item}_2 \cup \mathtt{Item}_3}$. Although $\mathrm{SAT}_{C_3}(\mathtt{Item})$ is a complicated expression involving several unions and minuses, itemsets satisfying the succinct constraint C_3 can be directly generate precisely (i.e., *without* generating and then excluding those itemset not satisfying C_3). More specifically, every itemset ν satisfying C_3 can be efficiently enumerated by combining (i) an item from $\mathtt{Item}_1$ (i.e., a *snack* item), (ii) an item from $\mathtt{Item}_2$ (i.e., a *soda* item), and (iii) some possible optional items from any of $\mathtt{Item}_1, \mathtt{Item}_2$ and $\mathtt{Item}_3$.

Although succinct constraints can be of various forms, they can be categorized into the following subclasses [9], depending on whether or not they are also anti-monotone:

- **SAM constraints.** Succinct anti-monotone constraints, such as $C_1 \equiv min(S.Qty) \geq 500$, $C_4 \equiv S.Price = 16$, $C_5 \equiv S.Type \subseteq \{beer, snack\}$ in Fig. 1; and
- **SUC constraints.** Succinct non-anti-monotone constraints, such as $C_2 \equiv max(S.Price) \geq 28$, $C_3 \equiv S.Type \supseteq \{snack, soda\}$, $C_6 \equiv soda \in S.Type$.

2.2 The FPS Algorithm

Like many FP-tree based algorithms, the FPS algorithm [9] consists of two main operations: (i) the construction of an FP-tree, and (ii) the growth of frequent patterns. The FPS algorithm first generates valid items (i.e., items satisfying succinct constraints) by using a member generating function, and then scans the transaction database to check the frequency of each valid item. As a result, valid frequent singleton itemsets can be found. The FPS algorithm then builds an initial FP-tree, which captures the content of the transaction database. By using this tree, a projected database can be formed for each valid item *X*. Here, an *X*-projected database is a collection of transactions having *X* as prefix. By *recursively* applying this FP-tree based mining process to each of these projected databases, valid frequent itemsets can be found. More specifically, suppose $X_1, X_2, X_3, \ldots$ are valid items. Then, itemsets containing X_1 can be computed from the $\{X_1\}$-projected database (and subsequent projected databases having X_1 as prefix), itemsets containing X_2 but not X_1 can be computed from the $\{X_2\}$-projected database, itemsets containing X_3 but not X_1 or X_2 can be computed from the $\{X_3\}$-projected database, and so on. Therefore, the entire mining process can be viewed as a divide-and-conquer approach of decomposing both the mining task and the transaction database according to the frequent patterns obtained so far. This leads to a focused search of smaller datasets.

The FPS algorithm consists of two components - **FPSam** and **FPSuc** - for handling SAM and SUC constraints, respectively. The key differences between these two components are as follows. *FPSam* divides the domain items into two groups (i.e., valid and invalid groups). Among the two groups, *only* items from the valid group are considered when forming the initial FP-tree and subsequent projected databases because any frequent itemset satisfying a SAM constraint is composed of only valid items. For example, any frequent itemset satisfying $C_1 \equiv min(S.Qty) \geq 500$ is composed of only items whose $Qty \geq 500$. In contrast, *FPSuc* divides the domain items into mandatory and optional groups[1]. It uses items from the mandatory and optional groups because any frequent itemset satisfying a SUC constraint is composed of mandatory items and possibly some optional items. For example, any frequent itemset satisfying $C_2 \equiv max(S.Price) \geq 28$ is composed of at least one item whose $Price \geq 28$ and possibly some optional items (whose *Prices* are unimportant). During the mining process, items are ordered in such a way that mandatory items appear before optional ones (i.e., all mandatory items appear below any optional ones in the FP-tree). Projected databases are formed only for the mandatory items, with the aforementioned item ordering. Consequently, all and only those frequent itemsets having appropriate mandatory items as prefix can be computed. In other words, all computed itemsets are valid (i.e., guaranteed to contain mandatory items and may contain some optional items). While more details can be found in the work of Leung et al. [9], we use the following example to illustrate an execution of the (static) FPS algorithm.

Example 1. Consider the following transaction database:

Transaction ID	Contents: *itemset*
t_1	$\{a, b, d\}$
t_2	$\{a, b, c, d, e, g\}$
t_3	$\{a, c\}$
t_4	$\{a, b, c, d, e, g\}$
t_5	$\{c\}$

with the auxiliary information from Fig. 1:

Item	a	b	c	d	e	f	g
Price	36	12	24	28	20	32	16
Qty	700	300	400	500	800	600	200
Type	*soda*	*soda*	*snack*	*beer*	*beer*	*beer*	*meat*

Let the minimum support threshold be 2 (i.e., 40%). The FPS algorithm first generates items satisfying the succinct constraint $C_2 \equiv max(S.Price) \geq 28$ by

[1] More precisely, FPSuc divides the domain items into k mandatory groups and one optional group. Although constraint $C_3 \equiv \{snack, soda\}$ uses two mandatory groups (one for *snack* items and another for *soda* items), *most of SUC constraints require only one mandatory group.* For lack of space, we focus on the latter (i.e., $k = 1$) in this paper.

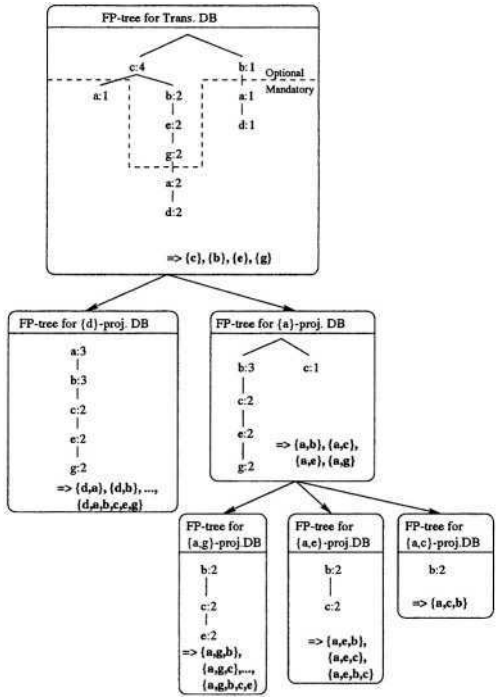

Fig. 2. The FPS algorithm mines frequent itemsets satisfying SUC constraint C_2

using a member generating function, and then scans the transaction database to check the frequency of each valid items. As a result, it finds valid frequent singleton itemsets $\{a\}$ and $\{d\}$. Then, FPS builds an initial FP-tree, as shown in Fig. 2. (The frequency of each item is shown in the figure, e.g., "c:4" in the tree indicates that the frequency of $\{c\}$ is 4.) Afterwards, FPS forms a projected database for each valid item (i.e., items a and d). This FP-tree based mining process is applied to the $\{d\}$-projected database to find all 31 valid frequent itemsets containing d. Similarly, the mining process is recursively applied to the $\{a\}$-projected database (and projected databases of the supersets of $\{a\}$). As a result, all the 15 valid frequent itemsets containing a but not d are found. □

3 Handling Dynamic Changes to Succinct Constraints

After reviewing the related work, let us start our discussion on the contribution of this paper – the development of the dyFPS algorithm. Like FPS, our dyFPS algorithm consists of two main components: *dyFPSam* and *dyFPSuc* for handling a dynamic change (e.g., a tightening change, a relaxing change) to a SAM constraint and a SUC constraint, respectively. Here, we assume that, at any point in time, there is at most one succinct constraint being modified. Of course, during the entire mining process, many different constraints can be changed. The

base case of our discussion below is on how to deal with changes to the constant *const*. When *const* is modified by the users in the direction of restricting the new solution space (denoted as V_{new}) to be a *subset* of the old space (denoted as V_{old}), we call this a *tightening change*. Otherwise, whenever the change to *const* corresponds to the situation where the new solution space contains the old space, we call this a *relaxing change*. For example, if the original succinct constraint is $max(S.Price) \geq 28$, then (i) changing the constant *const* from 28 to 32 corresponds to a tightening change and (ii) changing *const* from 28 to 24 corresponds to a relaxing change. Clearly, inserting a new constraint is a special case of a tightening change, and deleting an old constraint is an extreme case of a relaxing change. Thus, while our discussion below is confined to changing the constant *const*, any other modification (e.g., modifying $max(S.Price) \geq 28$ to $max(S.Price) \leq 28$) can be dealt with as a pair of constraint deletion and insertion.

A naïve approach of handling dynamic changes to succinct constraints is to simply ignore all valid frequent itemsets that have been produced so far with respect to the old constraint C_{old} (i.e., ignore all the "processed" itemsets) and rerun the FPS algorithm again with the new constraint C_{new}. Obviously, this approach can be costly because it does not reuse any "processed" frequent itemsets satisfying C_{old}.

Is there any better approach? When computing frequent itemsets satisfying C_{new}, can we reuse some of those "processed" frequent itemsets satisfying C_{old}? The answer to these questions is yes. In the remainder of this section, we show how our dyFPS algorithm pushes the constraints deep inside the mining process for effective pruning. The algorithm reuses the "processed" itemsets (i.e., valid frequent itemsets that have been produced with respect to C_{old}) as much as possible when mining valid frequent itemsets satisfying C_{new}.

3.1 Handling Dynamic Changes to a SAM Constraint

By definition, a **tightening change** from C_{old} to C_{new} corresponds to a restriction of the old solution space V_{old}. In other words, the new solution space V_{new} is a *subset* of V_{old}. To accommodate C_{new} dynamically, the dyFPS algorithm carries out two main operations as follows:

- For processed frequent itemsets satisfying C_{old}, check if they still satisfy C_{new}.
- For unprocessed itemsets, dyFPS only generates those satisfying C_{new}.

Recall that any frequent itemset ν satisfying a SAM constraint is composed of only valid items (i.e., items satisfying the constraint individually): $\nu \subseteq G^V$ (where G^V denotes a set of valid items). For a tightening change, any frequent itemset ν satisfying C_{new} is a subset of G^V_{new} (where G^V_{new} denotes the set of items satisfying C_{new}), which is a *subset* of G^V_{old} (where G^V_{old} denotes the set of items satisfying C_{old}):

$$\nu \subseteq G^V_{new} \subseteq G^V_{old}. \quad (1)$$

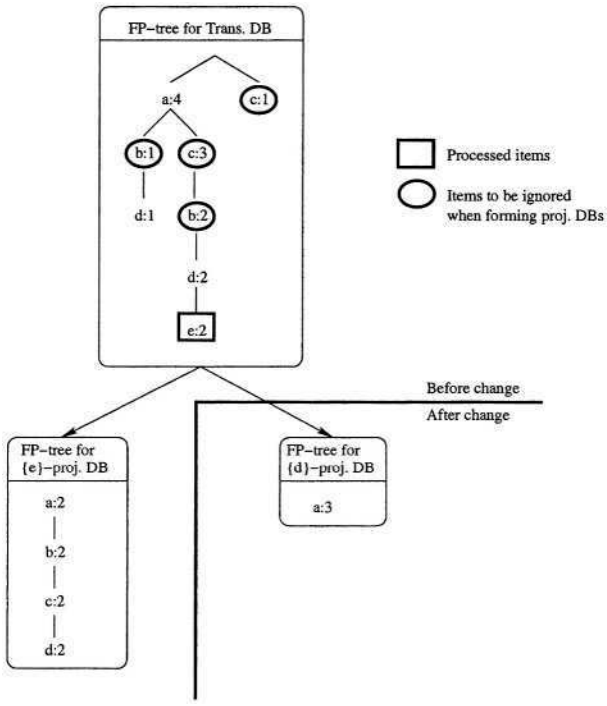

Fig. 3. Handling a dynamic tightening change to a SAM constraint using dyFPSam

Hence, after a SAM constraint is tightened, dyFPSam uses the existing FP-tree to generate the unprocessed itemsets satisfying C_{new}. Specifically, the algorithm forms projected databases *only* for unprocessed items satisfying C_{new}. When forming an $\{X\}$-projected database for an item $X \in G^V_{new}$, the dyFPSam algorithm *excludes* all **negative delta items** (i.e., the items that satisfy C_{old} but not C_{new}), which can be efficiently enumerated due to succinctness. As a result, dyFPSam can compute all the valid frequent itemsets (that have not been processed before the constraint change). To complete the mining process, dyFPSam needs to check the validity of all the frequent itemsets that were processed before the change (i.e., to check whether these itemsets still satisfy C_{new}). Example 2 shows how dyFPSam handles a tightening change to a SAM constraint.

Example 2. Consider the same transaction database & minimum support threshold as in Example 1. Suppose a SAM constraint $C_{old} \equiv min(S.Qty) \geq 300$ is tightened to $C_{new} \equiv min(S.Qty) \geq 500$ after the $\{e\}$-projected database has been processed (i.e., after dyFPSam computed all 15 itemsets containing e, such as $\{e, a\}, \{e, b\}, ..., \{e, a, b, c, d\}$). Then, due to succinctness, the negative delta items can be efficiently enumerated: items b and c. Therefore, as shown in Fig. 3, when forming the $\{d\}$-projected database, dyFPSam excludes items b and c from the extracted paths $\langle a, b\rangle$:1 and $\langle a, c, b\rangle$:2, and builds an FP-tree for

the $\{d\}$-projected database (which contains a:3). Consequently, a valid frequent itemset $\{d, a\}$ is computed. Finally, to complete the mining process, dyFPSam checks all the 15 itemsets containing e (i.e., the "processed" itemsets satisfying C_{old}), and finds that all of them satisfy C_{new}. □

We have discussed how dyFPSam handles a dynamic tightening change to a SAM constraint. In the remainder of this section, let us consider how dyFPSam handles a dynamic relaxing change to a SAM constraint. In general, a **relaxing change** from C_{old} to C_{new} has different and tougher computational requirements than a tightening change. The reason is that a tightening change leads to the situation where $V_{new} \subseteq V_{old}$; so, all that is needed is to verify whether every itemset ν satisfying C_{old} also satisfies C_{new}. In contrast, for a relaxing change, this verification is unnecessary because $V_{new} \supseteq V_{old}$. What is needed, however, is to compute all the "missing" itemsets (i.e., the itemsets that satisfy C_{new} but were not generated before the constraint was relaxed). Therefore, dyFPS needs to carry out the following operations:

- For processed itemsets satisfying C_{old}, no further constraint check is required because they also satisfy C_{new}.
- For unprocessed itemsets, dyFPS generates (i) the remaining itemsets satisfying C_{old} and (ii) those satisfying C_{new} but not C_{old}.

For a relaxing change, any frequent itemset ν satisfying C_{new} is a subset of G^V_{new}, which is a *superset* of G^V_{old}:

$$\nu \subseteq G^V_{new}, \quad \text{but } G^V_{new} \supseteq G^V_{old}. \tag{2}$$

Hence, after a SAM constraint is relaxed, dyFPSam adds **positive delta items** (i.e., items satisfying C_{new} but not C_{old}) to appropriate branches of the tree[2]. Then, the algorithm forms projected databases for all unprocessed items satisfying C_{new}. Again, due to succinctness, these items – as well as positive delta items – can be efficiently enumerated. The example below shows how dyFPSam handles a relaxing change to a SAM constraint.

Example 3. Consider the same transaction database & minimum support threshold as in Example 1. Suppose a SAM constraint $C_{old} \equiv min(S.Qty) \geq 500$ is relaxed to $C_{new} \equiv min(S.Qty) \geq 300$ after the $\{e\}$-projected database has been processed (i.e., after obtaining itemsets $\{e, a\}$, $\{e, d\}$ and $\{e, a, d\}$). Then, due to succinctness, the positive delta items can be efficiently enumerated: items b and c. They are added to the appropriate branches of the tree. Afterwards, as shown in Fig. 4, projected databases are formed for all unprocessed items satisfying C_{new} (e.g., for items d, b and c). Valid frequent itemsets can then be computed from these projected databases. □

[2] If I/O is a concern, one can keep both valid and invalid items in the tree. By so doing, positive delta items are already in the tree. This would save a database scan.

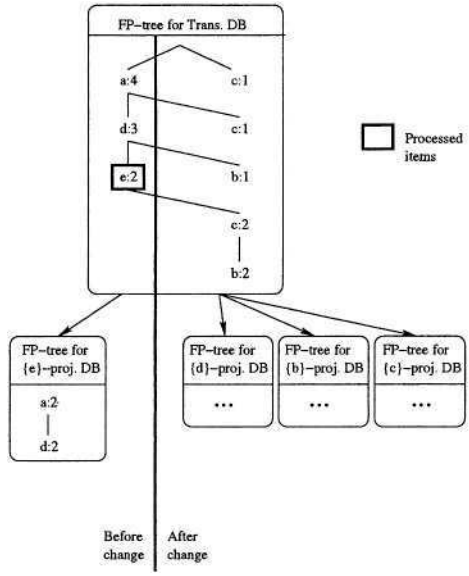

Fig. 4. Handling a dynamic relaxing change to a SAM constraint using dyFPSam

3.2 Handling Dynamic Changes to a SUC Constraint

One can observe from the previous section that dynamic changes to a SAM constraint cause the inclusion, or the exclusion, of items from the FP-tree. However, for dynamic changes to a SUC constraint, the situation is different. A reason is that, for a SUC constraint, its initial FP-tree contains not only mandatory items (i.e., items satisfying the constraint C) but also optional items (i.e., items not satisfying C). Changes to a SUC constraint only cause *a change of membership* (e.g., from the mandatory group G^M to the optional group G^O for the tightening change, and the opposite for the relaxing change).

Recall that any frequent itemset ν satisfying a SUC constraint is composed of at least one mandatory item and possibly some optional items:

$$\nu = \{X\} \cup \beta \cup \gamma \tag{3}$$

where (i) $X \in G^M$, (ii) $\beta \subseteq G^M$, and (iii) $\gamma \subseteq G^O$. So, after a SUC constraint is tightened, some of the mandatory items become optional (i.e., change their membership from mandatory to optional). These items are the ones that satisfy C_{old} but not C_{new}; we denote the group containing these items as $G^{M \to O}$, which is a subgroup of G^M. Similarly, we denote the subgroup of G^M that contains items satisfying both C_{new} and C_{old} (i.e., the unchanged group) as $G^{M \to M}$.

A complication of handling dynamic changes to a SUC constraint is that effective computation of itemsets satisfying a SUC constraint relies on the item ordering in the FP-tree. So far, we have only imposed the "inter-group" item ordering. All items from the mandatory group must come before any items from

the optional group (i.e., all mandatory items appear below any optional items in the FP-tree). However, there is no restriction imposed on the "intra-group" ordering (e.g., no restriction on the ordering of items within G^M). An item belonging to $G^{M \to M}$ may consistently appear above or below an item belonging to $G^{M \to O}$ in the FP-tree. If dyFPSuc were to apply the usual projection technique, the algorithm could be incomplete because it could possibly miss some itemsets containing an item in $G^{M \to O}$.

To ensure completeness, one could reorder the items so that items belonging to $G^{M \to O}$ appear after/above all the items belonging to $G^{M \to M}$. However, this approach may incur a costly overhead due to node split and/or node merge (i.e., the tree reorganization cost). The dyFPSuc algorithm avoids constructing a new FP-tree by using the existing FP-tree as follows. It forms projected databases *only* for items satisfying C_{new}. When dyFPSuc forms an $\{X\}$-projected database for an item *X,* in addition to *including* all items that are *above X* (as usual), dyFPSuc also *includes* all "unprocessed" items belonging to $G^{M \to O}$ that are *below X.* Due to succinctness, the items belonging to $G^{M \to O}$ can be efficiently enumerated. Note that this approach gives the same results as those produced by reordering items, but it avoids the costly overhead. As usual, for a tightening change, the algorithm needs to check the validity of all the frequent itemsets that were processed before the change (i.e., to check whether these "processed" itemsets still satisfy C_{new}). To gain a better understanding on how dyFPSuc handles a tightening change to a SUC constraint, let us consider the following example.

Example 4. Consider the same transaction database & minimum support threshold as in Example 1. Suppose a SUC constraint $C_{old} \equiv max(S.Price) \geq 20$ is tightened to $C_{new} \equiv max(S.Price) \geq 28$ after the $\{e\}$-projected database has been processed (i.e., after obtaining all 31 itemsets containing the mandatory item e, such as $\{e, a\}, \{e, b\}, ..., \{e, a, b, c, d, g\}$). Then, due to succinctness, items belonging to $G^{M \to O}$ can be efficiently enumerated: items c and e. Therefore, as shown in Fig. 5, when forming the $\{d\}$-projected database, dyFPSuc includes (i) all the items that are above d and (ii) all the "unprocessed" items belonging to $G^{M \to O}$ that are below d. Since item e has already been processed, it is not included. Then, dyFPSuc forms the $\{a\}$-projected database. Here, in addition to including all the items that are above a (i.e., items b and g), dyFPSuc also includes all the "unprocessed" items belonging to $G^{M \to O}$ that are below a (i.e., item c). Valid frequent itemsets can then be computed using these projected databases. Finally, to complete the mining process, dyFPSuc checks all the 31 itemsets containing e (i.e., the "processed" itemsets satisfying C_{old}), and finds that 24 of them satisfy C_{new}. □

A tightening change to a SUC constraint causes some items to change their membership from mandatory to optional. In contrast, a relaxing change to a SUC constraint leads to the opposite situation – that is, some items to change their membership from optional to mandatory.

Since both tightening and relaxing changes cause changes of membership, their treatment is quite similar, except the following. For a relaxing change, some optional items become mandatory; thus, dyFPSuc forms projected databases

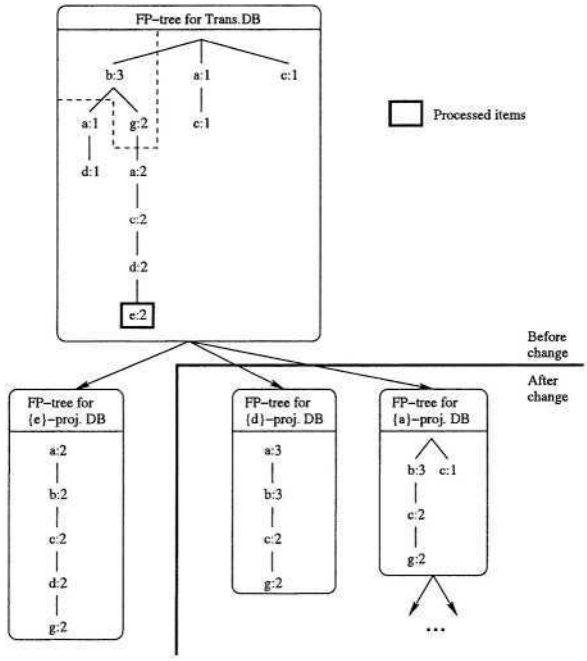

Fig. 5. Handling a dynamic tightening change to a SUC constraint using dyFPSuc

in such a way that it gives the same results as those produced by reordering items (i.e., reordering the items in such a way that mandatory items appear before/below all the optional items). For lack of space, we do not describe it further. We use Example 5 to illustrate how dyFPSuc handles a relaxing change to a SUC constraint.

Example 5. Consider the same transaction database & minimum support threshold as in Example 1. Suppose a SUC constraint $C_{old} \equiv max(S.Price) \geq 28$ is relaxed to $C_{new} \equiv max(S.Price) \geq 20$ after the $\{d\}$-projection has been processed (i.e., after obtaining all 31 itemsets containing d, such as $\{d, a\}, \{d, b\}, \{d, c\}, ...,$ $\{d, a, b, c, e, g\}$). Then, due to succinctness, the optional items (w.r.t. both C_{old} and C_{new}) can be efficiently enumerated: items b and g. Therefore, as shown in Fig. 6, dyFPSuc forms the $\{a\}$-projected database as usual. When forming the $\{e\}$-projected database, dyFPSuc includes all the items that are above e (i.e., items b and c) and all the "unprocessed" optional items that are below e (i.e., item g). Similarly, when dyFPSuc forms the $\{c\}$-projected database, dyFPSuc includes all the items that are above c (which is none) and all the "unprocessed" optional items that are below c (i.e., items b and g). Consequently, valid frequent itemsets can be computed by using these projected databases. □

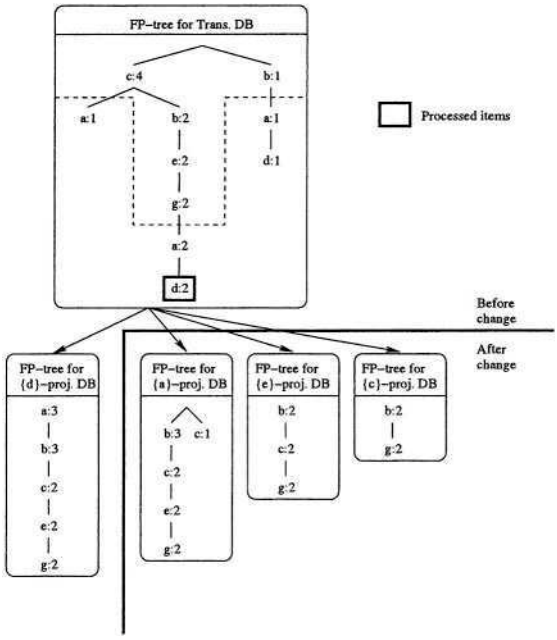

Fig. 6. Handling a dynamic relaxing change to a SUC constraint using dyFPSuc

4 Experimental Results

The experimental results cited below are based on a transaction database of 100k records with an average transaction length of 10 items, and a domain of 1000 items. The database was generated by the program developed at IBM Almaden Research Center [3]. Unless otherwise specified, we used a minimum support threshold of 0.01%. All experiments were run in a time-sharing environment in a 700 MHz machine. The reported figures are based on the average of multiple runs. In the experiment, we mainly compared two algorithms that were implemented in C: (i) **dyFPS** vs. (ii) **Rerun FPS.**

Recall that a naïve approach of handling dynamic changes to succinct constraints is to simply ignore all valid frequent itemsets that have been produced so far with respect to the old constraint C_{old} (i.e., ignore all "processed" itemsets) and to rerun the FPS algorithm again using the new constraint C_{new}. A better approach is to use our dyFPS algorithm, which reuses all "processed" itemsets. To evaluate the effectiveness of our algorithm, in this experiment, a SAM constraint $C_{old} \equiv max(S.Price) \leq 10$ is tightened to $C_{new} \equiv max(S.Price) \leq 8$. The percentage *old pct* of items having $Price \leq 10$ and the percentage *new pct* of items having $Price \leq 8$ are set in such a way that *new pct* = *old pct* – 20%. We varied *old pct* from 60% to 80%. The x-axis in Fig. 7(a) shows the percentage x of itemsets processed before tightening the constraint, and x varied from 10% to 90%. The y-axis shows the total runtime (in seconds) of both algorithms (dyFPS

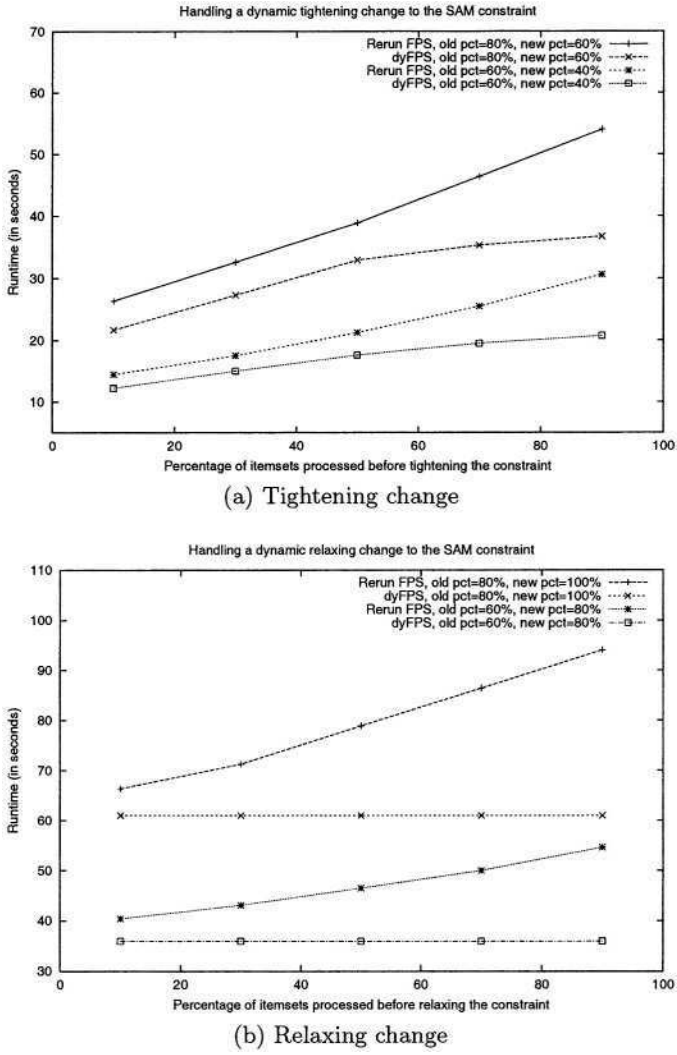

Fig. 7. Effectiveness of handling dynamic constraint changes: dyFPS vs. Rerun

and the Rerun FPS). From the graph, it is clear that our dyFPS algorithm always beats the rerun of FPS, but the extent varies under different situations. When x is high (i.e., more itemsets are processed), the relative speedup is high. The reason is that the Rerun FPS ignores all the itemsets produced w.r.t. C_{old} and (re-)computes itemsets satisfying C_{new}. Hence, when x is high, more itemsets are generated and ignored by the Rerun FPS. Out of these itemsets (produced using C_{old}), many of them satisfy C_{new}. The higher the percentage x, the higher is the number of processed itemsets being ignored! Our dyFPS algorithm, on the other hand, reuses the processed itemsets. Hence, when x increases, the number

of processed itemsets needed to be check at the final step of the mining process increases. However, the time required for checking the validity of these itemsets is much less than that required for (re-)computing itemsets. This explains why it is more beneficial to use dyFPS than to rerun FPS – especially when x *is* high.

While Figure 7(a) shows the results for a tightening change, Figure 7(b) shows the results for a relaxing change. Specifically, we relaxed a SAM constraint $C_{old} \equiv max(S.Price) \leq 10$ to $C_{new} \equiv max(S.Price) \leq 12$. The percentage *old pct* of items having $Price \leq 10$ and the percentage *new pct* of items having $Price \leq 12$ are set in such a way that *new pct* = *old pct* + 20%. We varied *old pct* from 60% to 80%. Again, it is clear from the graph that our dyFPS algorithm always beats the rerun of FPS. Similar to the tightening case, the Rerun FPS ignores all the itemsets produced w.r.t. C_{old} and (re-)computes itemsets satisfying C_{new}. Hence, the higher the percentage x, the higher is the number of itemsets that are generated and ignored by the Rerun FPS. Here, *all* of these itemsets (produced using C_{old}) satisfy C_{new} because $V_{new} \supseteq V_{old}$ for the relaxing change. Thus, using the Rerun FPS is really a waste of computation! On the other hand, our dyFPS algorithm reuses all these processed itemsets. After the constraint C_{old} is relaxed to become C_{new}, our dyFPS algorithm generates (i) the remaining itemsets satisfying C_{old} and (ii) those satisfying C_{new} but not C_{old}. This explains why the runtime of dyFPS is quite steady. Therefore, it is more beneficial to use dyFPS than to rerun FPS – especially when x is high.

We have also experimented with the following cases: (i) tightened a SUC constraint $C_{old} \equiv min(S.Price) \leq 10$ to $C_{new} \equiv min(S.Price) \leq 8$, and (ii) relaxed a SUC constraint $C_{old} \equiv min(S.Price) \leq 10$ to $C_{new} \equiv min(S.Price) \leq 12$. The results produced are consistent with those for the SAM constraint. Our proposed dyFPS algorithm outperforms the rerun of FPS. In summary, these experimental results show the effectiveness of exploiting succinctness properties of the constraints (e.g., reusing the "processed" itemsets).

5 Conclusions

A key contribution of this paper is to optimize the performance of, and to increase functionality of, a dynamic FP-tree based constrained mining algorithm. To this end, we proposed (and studied) the novel algorithm of dyFPS. The algorithm efficiently handles dynamic changes to succinct constraints, and effectively exploits succinctness properties. As a result, the constraints are pushed deep inside the mining process, and thereby avoiding algorithm rerun and leading to effective pruning.

In ongoing work, we are interested in exploring improvements to the dyFPS algorithm. For example, we are interested in investigating effective technique for handling dynamic changes to some more sophisticated Boolean combinations of succinct constraints. Along this direction, an interesting question to explore is how to extend our dyFPS algorithm to mine frequent patterns satisfying non-succinct constraints.

Acknowledgement

This project is partially sponsored by The University of Manitoba in the form of research grants.

References

1. Agrawal, R., Imielinski, T., Swami, A.: Mining association rules between sets of items in large databases. In: Proc. SIGMOD 1993. 207–216
2. Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., Verkamo, A.I.: Fast discovery of association rules. In: Advances in Knowledge Discovery and Data Mining. AAAI/MIT Press (1996) chapter 12
3. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules. In: Proc. VLDB 1994. 487–499
4. Bayardo, R.J., Agrawal, R., Gunopulos, D.: Constraint-based rule mining in large, dense databases. In: Proc. ICDE 1999. 188–197
5. Garofalakis, M.N., Rastogi, R., Shim, K.: SPIRIT: sequential pattern mining with regular expression constraints. In: Proc. VLDB 1999. 223–234
6. Grahne, G., Lakshmanan, L.V.S., Wang, X.: Efficient mining of constrained correlated sets. In: Proc. ICDE 2000. 512–521
7. Han, J., Pei, J., Yin, Y.: Mining frequent patterns without candidate generation. In: Proc. SIGMOD 2000. 1–12
8. Lakshmanan, L.V.S., Ng, R., Han, J., Pang, A.: Optimization of constrained frequent set queries with 2-variable constraints. In: Proc. SIGMOD 1999. 157–168
9. Leung, C.K.-S., Lakshmanan, L.V.S., Ng, R.T.: Exploiting succinct constraints using FP-trees. SIGKDD Explorations **4**(1) (June 2002) 40–49
10. Ng, R.T., Lakshmanan, L.V.S., Han, J., Pang, A.: Exploratory mining and pruning optimizations of constrained associations rules. In: Proc. SIGMOD 1998. 13–24
11. Pei, J., Han, J., Lakshmanan, L.V.S.: Mining frequent itemsets with convertible constraints. In: Proc. ICDE 2001. 433–442
12. Srikant, R., Vu, Q., Agrawal, R.: Mining association rules with item constraints. In: Proc. KDD 1997. 67–73

Semantic Optimization of Preference Queries*

Jan Chomicki

Dept. of Computer Science and Engineering
University at Buffalo
Buffalo, NY 14260-2000
chomicki@cse.buffalo.edu

Abstract. Preference queries are relational algebra or SQL queries that contain occurrences of the winnow operator (*find the most preferred tuples in a given relation*). We present here a number of semantic optimization techniques applicable to preference queries. The techniques make it possible to remove redundant occurrences of the winnow operator and to apply a more efficient algorithm for the computation of winnow. We also study the propagation of integrity constraints in the result of the winnow. We have identified necessary and sufficient conditions for the applicability of our techniques, and formulated those conditions as *constraint satisfiability* problems.

1 Introduction

The notion of *preference* is becoming more and more ubiquitous in present-day information systems. Preferences are primarily used to filter and personalize the information reaching the users of such systems. In database systems, preferences are usually captured as *preference relations* that are used to build *preference queries* [7,8,17,18]. From a formal point of view, preference relations are simply binary relations defined on query answers. Such relations provide an abstract, generic way to talk about a variety of concepts like priority, importance, relevance, timeliness, reliability etc. Preference relations can be defined using logical formulas [7,8] or special preference constructors [17] (preference constructors can be expressed using logical formulas). The embedding of preference relations into relational query languages is typically provided through a relational operator that selects from its argument relation the set of the *most preferred tuples,* according to a given preference relation. This operator has been variously called *winnow* (the term we use here) [7,8], BMO [17], and Best [28]. (It is also implicit in skyline queries [3].) Being a relational operator, winnow can clearly be combined with other relational operators, in order to express complex preference queries.

* Research supported by NSF Grant IIS-0307434.

B. Kuijpers and P. Revesz (Eds.): CDB 2004, LNCS 3074, pp. 128–142, 2004.

Example 1. We introduce an example used throughout the paper. Consider the relation *Book*(*ISBN, Vendor, Price*) and the following preference relation $\succ_{C_1}$ between *Book* tuples:

> *prefer one Book tuple to another if and only if their ISBNs are the same and the Price of the first is lower.*

Consider the instance r_1 of *Book* in Figure 1. Then the winnow operator ω_{C_1} returns the set of tuples in Figure 2.

ISBN	*Vendor*	*Price*
0679726691	BooksForLess	$14.75
0679726691	LowestPrices	$13.50
0679726691	QualityBooks	$18.80
0062059041	BooksForLess	$7.30
0374164770	LowestPrices	$21.88

Fig. 1. The Book relation

ISBN	*Vendor*	*Price*
0679726691	LowestPrices	$13.50
0062059041	BooksForLess	$7.30
0374164770	LowestPrices	$21.88

Fig. 2. The result of winnow

Example 2. The above example is a one-dimensional skyline query. To see an example of a two-dimensional skyline, consider the schema of *Book* expanded by another attribute *Rating.* Define the following preference relation C_2:

> *prefer one Book tuple to another if and only if their ISBNs are the same and the Price of the first is lower and the Rating of the first is not lower, or the Price of the first is not higher and the Rating of the first is higher.*

Then ω_{C_2} is equivalent to the following skyline (in the terminology of [3]):

```
SKYLINE ISBN DIFF, Price MIN, Rating MAX.
```

The above notation indicates that only books with the same ISBN should be compared, that Price should be minimized, and Rating maximized. In fact, the tuples in the skyline satisfy the property of *Pareto-optimality,* well known in economics.

Preference queries can be reformulated in relational algebra or SQL, and thus optimized and evaluated using standard relational techniques. However, it has been recognized that specialized evaluation and optimization techniques promise in this context performance improvements that are otherwise unavailable. A number of new algorithms for the evaluation of skyline queries (a special class of preference queries) have been proposed [3,4,23,26]. Some of them can be used to evaluate general preference queries [8]. Also, algebraic laws that characterize the interaction of winnow with the standard operators of relational algebra have been formulated [8,15,16]. Such laws provide a foundation for the rewriting of preference queries. For instance, necessary and sufficient conditions for pushing a selection through winnow are described in [8]. The algebraic laws cannot be applied unconditionally. In fact, the preconditions of their applications refer to the *validity* of certain *constraint formulas.*

In this paper, we pursue the line of research from [8] a bit further. We study *semantic optimization* of preference queries. Semantic query optimization has been extensively studied for relational and deductive databases [6]. As a result, a body of techniques dealing with specific query transformations like join elimination and introduction, predicate introduction etc. has been developed. We view semantic query optimization very broadly and classify as *semantic* any query optimization technique that makes use of integrity constraints. In the context of preference queries, we focus on the winnow operator. Despite the presence of specialized evaluation techniques, winnow is still quite an expensive operation. We develop optimizing techniques that:

1. remove redundant occurrences of winnow;
2. recognize when more efficient evaluation of winnow is possible.

More efficient evaluation of winnow can be achieved, for example, if the given preference relation is a *weak order* (a negatively transitive strict partial order). We show that even when the preference relation is not a weak order (as in Example 1), it may become equivalent to a weak order on the relations satisfying certain integrity constraints. We show a very simple, single-pass algorithm for evaluating winnow under those conditions. We also pay attention to the issue of satisfaction of integrity constraints in the result of applying winnow. In fact, some constraints may hold in the result of winnow, even though they do not hold in the relation to which winnow is applied. Combined with known results about the preservation of integrity constraints by relational algebra operators [21,22], our results provide a way for optimizing not only single occurrences of winnow but also complex preference queries. As in the case of the algebraic transformations described in [8], the semantic transformations described in this paper have preconditions referring to the validity of certain constraint formulas. Thus, such preconditions can be checked using well established constraint satisfaction techniques [12][1].

The plan of the paper is as follows. In Section 2 we define basic notions. We limit ourselves here to integrity constraints that are *functional dependencies.* In Section 3 we address the issue of eliminating redundant occurrences of winnow. In Section 4 we study weak orders. In Section 5 we characterize dependencies holding in the result of winnow. In Section 6 we show how our results can be generalized to *constraint-generating dependencies* [2]. We briefly discuss related work in Section 7 and conclude in Section 8.

2 Basic Notions

We are working in the context of the relational model of data. For concreteness, we consider two infinite domains: $\mathcal{D}$ (uninterpreted constants) and $\mathcal{Q}$ (rational numbers). Other domains could be considered as well without influencing most of the results of the paper. We assume that database instances are finite. Additionally, we have the standard built-in predicates.

[1] A formula is valid iff its negation is unsatisfiable.

2.1 Preference Relations

Definition 1. *Given a relation schema* $R(A_1 \cdots A_k)$ *such that* U_i, $1 \leq i \leq k$, *is the domain (either* $\mathcal{D}$ *or* $\mathcal{Q}$*) of the attribute* A_i, *a relation* $\succ$ *is a* preference relation over R *if it is a subset of* $(U_1 \times \cdots \times U_k) \times (U_1 \times \cdots \times U_k)$.

Intuitively, $\succ$ will be a binary relation between tuples from the same (database) relation. We say that a tuple t_1 *dominates* a tuple t_2 in $\succ$ if $t_1 \succ t_2$.

Typical properties of the relation $\succ$ include:

- *irreflexivity:* $\forall x.\ x \not\succ x$,
- *asymmetry:* $\forall x, y.\ x \succ y \Rightarrow y \not\succ x$,
- *transitivity:* $\forall x, y, z.\ (x \succ y \wedge y \succ z) \Rightarrow x \succ z$,
- *negative transitivity:* $\forall x, y, z.\ (x \not\succ y \wedge y \not\succ z) \Rightarrow x \not\succ z$,
- *connectivity:* $\forall x, y.\ x \succ y \vee y \succ x \vee x = y$.

The relation $\succ$ is:

- a *strict partial order* if it is irreflexive and transitive (thus also asymmetric);
- a *weak order* if it is a negatively transitive strict partial order;
- a *total order* if it is a connected strict partial order.

At this point, we do not assume any properties of $\succ$, although in most applications it will satisfy at least the properties of *strict partial order.*

Definition 2. *A* preference formula (pf) $C(t_1, t_2)$ *is a first-order formula defining a preference relation* $\succ_C$ *in the standard sense, namely*

$$t_1 \succ_C t_2 \text{ iff } C(t_1, t_2).$$

An intrinsic preference formula (ipf) *is a preference formula that uses only built-in predicates.*

We will limit our attention to preference relations defined using intrinsic preference formulas.

Because we consider two specific domains, $\mathcal{D}$ and $\mathcal{Q}$, we will have two kinds of variables, $\mathcal{D}$-variables and $\mathcal{Q}$-variables, and two kinds of atomic formulas:

- *equality constraints:* $x = y$, $x \neq y$, $x = c$, or $x \neq c$, where x and y are $\mathcal{D}$-variables, and c is an uninterpreted constant;
- *rational-order constraints:* $x\theta y$ or $x\theta c$, where $\theta \in \{=, \neq, <, >, \leq, \geq\}$, x and y are $\mathcal{Q}$-variables, and c is a rational number.

Without loss of generality, we will assume that ipfs are in DNF (Disjunctive Normal Form) and quantifier-free (the theories involving the above domains admit quantifier elimination). We also assume that atomic formulas are closed under negation (also satisfied by the above theories). An ipf whose all atomic formulas are equality (resp. rational-order) constraints will be called an *equality* (resp. *rational-order*) ipf. Clearly, ipfs are a special case of general constraints [20], and define *fixed,* although possibly infinite, relations. By using the notation $\succ_C$ for a preference relation, we assume that there is an underlying preference formula C.

Definition 3. *Given an instance r of R and a preference relation $\succ_C$ over R, the* restriction $\succ_C|_r$ *of $\succ_C$ to r is defined as*

$$\succ_C|_r = \succ_C \cap\, r \times r.$$

2.2 Winnow

We define now an algebraic operator that picks from a given relation the set of the *most preferred tuples,* according to a given preference formula.

Definition 4. *If R is a relation schema and C a preference formula defining a preference relation $\succ_C$ over R, then the* winnow operator *is written as $\omega_C(R)$, and for every instance r of R:*

$$\omega_C(r) = \{t \in r \mid \neg\exists t' \in r.\ t' \succ_C t\}.$$

A preference query is a relational algebra query containing at least one occurrence of the winnow operator.

Example 3. Consider the relation *Book(ISBN,Vendor,Price)* (Example 1). The preference relation $\succ_{C_1}$ from this example can be defined using the formula C_1:

$$(i, v, p) \succ_{C_1} (i', v', p') \equiv i = i' \wedge p < p'.$$

The answer to the preference query $\omega_{C_1}(Book)$ provides for every book the information about the vendors offering the lowest price for that book.

2.3 Indifference

Every preference relation $\succ_C$ generates an indifference relation $\sim_C$: two tuples t_1 and t_2 are *indifferent* ($t_1 \sim_C t_2$) if neither is preferred to the other one, i.e., $t_1 \not\succ_C t_2$ and $t_2 \not\succ_C t_1$.

Proposition 1. *For every preference relation $\succ_C$, every relation r and every tuple $t_1, t_2 \in \omega_C(r)$, we have $t_1 = t_2$ or $t_1 \sim_C t_2$.*

2.4 Functional Dependencies

We assume that we are working in the context of a single relation schema R and all the integrity constraints are over that schema. The set of all instances of R satisfying a set of integrity constraints F is denoted as *Sat(F).* We say that F *entails* an integrity constraint f if every instance satisfying F also satisfies f.

A functional dependency (FD) $f \equiv X \rightarrow Y$, where X and Y are sets of attributes of R can be written down as the following logic formula:

$$\forall t_1 . \forall t_2.\ [R(t_1) \wedge R(t_2) \wedge t_1[X] = t_2[X]] \Rightarrow t_1[Y] = t_2[Y].$$

We use the following notation:

$$\varphi_f(t_1, t_2) \equiv t_1[X] = t_2[X] \Rightarrow t_1[Y] = t_2[Y].$$

For a set of FDs F, we define

$$\varphi_F \equiv \bigwedge_{f \in F} \varphi_f.$$

The *arity* of an FD $f \equiv X \rightarrow Y$ is the cardinality $|X \cup Y|$ of the set of attributes $X \cup Y$. The *arity* of a set of FDs F is the maximum arity of any FD in F.

Note that the set of attributes X in $X \rightarrow Y$ may be empty, meaning that each attribute in Y can assume only a single value.

3 Eliminating Redundant Occurrences of Winnow

Given an instance r of R, the operator ω_C is redundant if $\omega_C(r) = r$. If we consider the class of all instances of R, then such an operator is redundant for every instance iff $\succ_C$ is an empty relation. The latter holds iff C is unsatisfiable. However, we are interested only in the instances satisfying a given set of integrity constraints. Therefore, we will check whether the restriction $\succ_C|_r$ is empty for every instance r satisfying the given set of integrity constraints.

Definition 5. *Given a set of integrity constraints F, the operator ω_C is* redundant w.r.t. a set of integrity constraints F *if* $\forall r \in Sat(F)$, $\omega_C(r) = r$.

Theorem 1. *ω_C is redundant w.r.t. a set of FDs F iff the following formula is unsatisfiable:*

$$\varphi_F(t_1, t_2) \wedge t_1 \succ_C t_2$$

Proof. Assume that formula in the theorem is satisfiable. Then there are tuples t_a and t_b such that $\varphi_F(t_a, t_b)$ and $t_a \succ_C t_b$. Thus $t_b \notin \omega_C(\{t_a, t_b\})$ and thus ω_C is not redundant w.r.t. F. For the other direction, assume ω_C is not redundant w.r.t. F. Then there is an instance $r_0 \in Sat(F)$ and a tuple $t_b \in r_0$ such that $t_b \notin \omega_C(r_0)$. Thus, there must be a tuple t_a in r_0 such that $t_a \succ_c t_b$. Clearly, $\varphi_F(t_a, t_b)$ and therefore the formula in the theorem is satisfiable.

Theorem 1 shows that checking for redundancy w.r.t. a set of FDs F is a *constraint satisfiability* problem.

Example 4. Consider Example 3 in which the FD *ISBN* $\rightarrow$ *Price* holds. Then

$$\varphi_F \equiv i_1 = i_2 \Rightarrow p_1 = p_2$$

and $\varphi_F(t_1, t_2) \wedge t_1 \succ_{C_1} t_2$ is

$$(i_1 = i_2 \Rightarrow p_1 = p_2) \wedge i_1 = i_2 \wedge p_1 < p_2.$$

The last formula is clearly unsatisfiable, and thus the implication in Theorem 1 holds and we can infer that ω_{C_1} is redundant w.r.t. *ISBN* $\rightarrow$ *Price*.

How hard is it to check for redundancy w.r.t. a set of FDs F? We assume that the size of a preference formula C (over a relation R) in DNF is characterized by two parameters: *width*(C) – the number of disjuncts in C, and *span*(C) – the maximum number of conjuncts in a disjunct of C. Namely, if $C = D_1 \vee \cdots \vee D_m$, and each $D_i = C_{i,1} \wedge \cdots C_{i,k_i}$, then $width(C) = m$ and $span(C) = \max\{k_1, \ldots, k_m\}$.

Theorem 2. *If:*

- *the cardinality of the set of FDs F is $|F|$ and its arity is at most k;*
- *the given preference relation is defined using an ipf C containing only atomic constraints over the same domain and such that $width(C) \leq m$, $span(C) \leq n$;*
- *the time complexity of checking satisfiability of a conjunctive ipf with n conjuncts is in $O(T(n))$,*

then the time complexity of checking ω_C for redundancy with respect to F is in $O(m\ k^{k|F|}\ T(\max(k|F|, n)))$.

The paper [12] contains several results about checking satisfiability of conjunctive formulas. For instance, in the case of rational-order formulas, this problem is shown to be solvable in $O(n)$. This implies, for example, the following corollary.

Corollary 1. *If a preference relation is defined by a conjunctive rational-order ipf ($m = 1$) and the arity of F is at most 2, then checking ω_C for redundancy w.r.t. F can be done in time $O(n\ 2^{|F|})$.*

An analogous result can be derived for equality formulas. From now on we will only present detailed complexity analysis for rational-order formulas.

4 Weak Orders

We have defined weak orders as negatively transitive strict partial orders. Equivalently, they can be defined as strict partial orders for which the indifference relation is transitive. Intuitively, a weak order consists of a number (perhaps infinite) of linearly ordered layers. In each layer, all the elements are mutually indifferent and they are all above all the elements in lower layers.

Example 5. In the preference relation $\succ_{C_1}$ in Example 3, the first, second and third tuples are indifferent with the fourth and fifth tuples. However, the first tuple is preferred to the second, violating the transitivity of indifference. Therefore, the preference relation $\succ_{C_1}$ is not a weak order.

Example 6. A preference relation $\succ_{C_f}$, defined as

$$x \succ_{C_f} y \equiv f(x) > f(y)$$

for some real-valued function f, is a weak order but not necessarily a total order.

4.1 Computing Winnow

Many algorithms for evaluating winnow are possible. However, we discuss here those that have a good *blocking* behavior and thus are capable of efficiently processing very large data sets.

We first review BNL (Figure 3), a basic algorithm for evaluating winnow, and then show that for preference relations that are weak orders a much simpler and more efficient algorithm is possible. BNL was proposed in [3] in the context of *skyline queries*. However, [3] also noted that the algorithm requires only the properties of strict partial orders. BNL uses a fixed amount of main memory (a *window*). It also needs a temporary table for the tuples whose status cannot be determined in the current pass, because the available amount of main memory is limited.

1. clear the window W and the temporary table F;
2. make r the input;
3. repeat the following until the input is empty:
   (a) for every tuple t in the input:
   - t is dominated by a tuple in $W \Rightarrow$ ignore t,
   - t dominates some tuples in $W \Rightarrow$ eliminate the dominated tuples and insert t into W,
   - if t and all tuples in W are mutually indifferent $\Rightarrow$ insert t into W (if there is room), otherwise add t to F;

   (b) output the tuples from W that were added there when F was empty,
   (c) make F the input, clear the temporary table.

Fig. 3. BNL: Blocked Nested Loops

BNL keeps in the window the best tuples discovered so far (some of them may also be in the temporary table). All the tuples in the window are mutually indifferent and they all need to be kept, since each may turn out to dominate some input tuple arriving later. For weak orders, however, if a tuple t_1 dominates t_2, then any tuple indifferent to t_1 will also dominate t_2. In this case, indifference is an equivalence relation, and thus it is enough to keep in main memory only a single tuple *top* from the top equivalence class. In addition, one has to keep track of all members of that class (called the *current bucket B*), since they may have to be returned as the result of the winnow. The new algorithm WWO (Winnow for Weak Orders) is shown in Figure 4.

It is clear that WWO requires only a single pass over the input. It uses additional memory (whose size is at most equal to the size of the input) to keep track of the current bucket. However, this memory is only written and read once, the latter at the end of the execution of the algorithm. Clearly, for weak orders WWO is considerably more efficient than BNL. Note that for weak orders BNL does not simply reduce to WWO. Note also that if additional memory is not available, WWO can execute in a small, fixed amount of memory by using two

1. top := the first input tuple
2. $B := \{top\}$
3. for every subsequent tuple t in the input:
   - t is dominated by $top \Rightarrow$ ignore t,
   - t dominates $top \Rightarrow top := t$; $B := \{t\}$
   - t and top are indifferent $\Rightarrow B := B \cup \{t\}$
4. output B

Fig. 4. WWO: Weak Order Winnow

passes over the input: in the first, a top tuple is identified, and in the second, all the tuples indifferent to it are selected.

In [4] we proposed SFS, a more efficient variant of BNL for skyline queries, in which a presorting step is used. Because sorting may require more than one pass over the input, that approach will also be less efficient than WWO for weak orders (unless the input is already sorted).

4.2 Relative Weak Orders

Even if a preference relation $\succ_C$ is not a weak order in general, its restriction to a specific instance or a class of instances may be a weak order, and thus WWO may be applied to the computation of winnow. Again, we are going to consider the class of instances *Sat(F)* for a set of integrity constraints *F*.

Definition 6. *A preference relation $\succ_C$ is a* weak order relative to a set of integrity constraints *F if* $\forall r \in Sat(F)$, $\succ_C|_r$ *is a weak order.*

Theorem 3. *An irreflexive preference relation $\succ_C$ is a weak order relative to a set of FDs F iff the following formula is unsatisfiable:*

$$\varphi_F(t_1, t_2) \wedge \varphi_F(t_2, t_3) \wedge \varphi_F(t_1, t_3) \wedge t_1 \succ_C t_2 \wedge t_1 \sim_C t_3 \wedge t_2 \sim_C t_3.$$

Example 7. Consider Example 3, this time with the 0-ary FD $\emptyset \Rightarrow ISBN$. (Such a dependency might hold, for example, in a relation resulting from the selection $\sigma_{ISBN=c}$ for some constant c.) Note that

$$(i, v, p) \sim_c (i', v', p') \equiv i \neq i' \vee p = p'.$$

We construct the following formula, according to Theorem 3:

$$i_1 = i_2 \wedge i_2 = i_3 \wedge i_1 = i_3 \wedge i_1 = i_2 \wedge p_1 < p_2 \wedge (i_1 \neq i_3 \vee p_1 = p_3) \wedge (i_2 \neq i_3 \vee p_2 = p_3)$$

which is unsatisfiable. Therefore, $\succ_{C_1}$ is a weak order relative to the FD $\emptyset \Rightarrow ISBN$, and for every instance r satisfying this dependency, $\omega_{C_1}(r)$ can be computed using the single-pass algorithm WWO.

Theorem 4. *If:*

- *the cardinality of the set of FDs F is $|F|$ and its arity is at most k;*
- *the given preference relation is defined using an ipf C containing only atomic constraints over the same domain and such that $width(C) \leq m$, $span(C) \leq n$;*
- *the time complexity of checking satisfiability of a conjunctive ipf with n conjuncts is in $O(T(n))$,*

then the time complexity of checking whether $\succ_C$ is a weak order relative to F is in $O(m\ n^{4m}\ k^{k|F|}\ T(\max(k|F|, m, n)))$.

Corollary 2. *If a preference relation is defined by a conjunctive rational-order ipf ($m = 1$) and the arity of F is at most 2, then then the time complexity of checking whether $\succ_C$ is a weak order relative to F is in $O(n^5\ 2^{|F|})$.*

5 Propagation of Integrity Constraints

The study of propagation of integrity constraints by relational operators is essential for semantic optimization of complex queries. We need to know which integrity constraints hold in the results of such operators. The winnow operator returns a subset of a given relation, thus it preserves all the functional dependencies holding in the relation. However, we also know that winnow returns a set of tuples which are mutually indifferent. This property can be used to derive *new* dependencies that hold in the result of winnow without necessarily holding in the input relation. (New dependencies can also be derived for other relational operators, for example selection, as in Example 7.)

Theorem 5. *Let f be an FD and $\succ_C$ an irreflexive preference relation over R. The following formula*

$$t_1 \sim_C t_2 \wedge \neg\varphi_f(t_1, t_2)$$

is unsatisfiable iff for every instance r of R, $\omega_C(r)$ satisfies f.

Proof. We will call the FDs satisfying the condition in Theorem 5 *generated* by $\succ_C$ and denote the set of all such dependencies by G_C. It is easy to show that G_C is closed w.r.t. FD implication. Assume $f \notin G_C$. Then the formula in the theorem is satisfiable. Assume it is satisfied by tuples t_a and t_b ($t_a \neq t_b$ because otherwise $\neg\varphi(t_a, t_b)$ is false). Thus $r_0 = \{t_a, t_b\} \notin Sat(f)$. But $t_a \sim_C t_b$, $t_a \not\succ_C t_a$, and $t_b \not\succ_C t_b$. Thus $r_0 = \omega_C(r_0) \notin Sat(f)$.

In the other direction, assume that there is an instance r_0 such that $\omega_C(r_0) \notin Sat(f)$. By the properties of FDs, we can assume that $\omega_C(r_0)$ consists of two distinct tuples t_a and t_b. By Proposition 1, we know that $t_a \sim_C t_b$. Thus the formula is satisfied by t_a and t_b.

Example 8. Consider Example 3. Then the formula from Theorem 5 is

$$(i_1 \neq i_2 \vee p_1 = p_2) \wedge i_1 = i_2 \wedge p_1 \neq p_2$$

which is clearly unsatisfiable. Thus, the FD $ISBN \rightarrow Price$ holds in the result of ω_{C_1}, even though it might not hold in the input relation.

Theorem 6. *If:*

- *the arity of* f *is* k*;*
- *the given preference relation is defined using an ipf* C *containing only atomic constraints over the same domain and such that* $width(C) \leq m$*,* $span(C) \leq n$*;*
- *the time complexity of checking satisfiability of a conjunctive ipf with* n *conjuncts is in* $O(T(n))$*,*

then the time complexity of checking checking the condition in Theorem 5 is in $O(kn^{2m}\, T(\max(k, m)))$.

Corollary 3. *If a preference relation is defined by a conjunctive rational-order ipf* $(m = 1)$ *and the arity of* f *is at most* 2*, then the time complexity of checking the condition in Theorem 5 is in* $O(n^2)$.

6 Constraint-Generating Dependencies

Functional dependencies are a special case of *constraint-generating dependencies*[2].

Definition 7. *A* constraint-generating dependency (CGD) *can be expressed a formula of the following form:*

$$\forall t_1 \ldots \forall t_n.\ [R(t_1) \wedge \cdots \wedge R(t_n) \wedge \gamma(t_1, \ldots t_n)] \Rightarrow \gamma'(t_1, \ldots t_n)$$

where $\gamma(t_1, \ldots t_n)$ *and* $\gamma'(t_1, \ldots t_n)$ *are constraints over some constraint theory.*

CGDs are equivalent to denial constraints.

Example 9. We give here some examples of CGDs. Consider the relation *Emp* with attributes *Name, Salary,* and *Manager,* with *Name* being the primary key. The constraint that *no employee can have a salary greater that that of her manager* is a CGD:

$$\forall n, s, m, s', m'.\ [Emp(n, s, m) \wedge Emp(m, s', m')] \Rightarrow s \leq s'.$$

Similarly, single-tuple constraints (`CHECK` constraints in SQL2) are a special case of CGDs. For example, the constraint that *no employee can have a salary over $200000* is expressed as:

$$\forall n, s, m.\ Emp(n, s, m) \Rightarrow s \leq 200000].$$

It turns out that the problems studied in the present paper can be viewed as specific instances of the *entailment* (implication) of CGDs. To see that, let's define two special CGDs d_2^C and d_3^C for a given preference relation $\succ_C$ (and the corresponding indifference relation $\sim_C$):

$$d_2^C \equiv \forall t_1 . \forall t_2.\ R(t_1) \wedge R(t_2) \Rightarrow t_1 \sim_C t_2$$

and

$$d_3^C \equiv \forall t_1.\forall t_2.\forall t_3.\ \ R(t_1) \wedge R(t_2) \wedge R(t_3) \Rightarrow \neg(t_1 \succ_C t_2 \wedge t_1 \sim_C t_3 \wedge t_2 \sim_C t_3).$$

Then we have the following properties that generalize Theorems 1, 3, and 5.

Theorem 7. *ω_C is redundant w. r. t. a set of CGDs F iff F entails d_2^C.*

Theorem 8. *If $\succ_C$ is irreflexive, then $\succ_C$ is a weak order relative to a set of CGDs F iff F entails d_3^C.*

Theorem 9. *If $\succ_C$ is irreflexive, then a CGD f is entailed by d_2^C iff for every instance r of R, $\omega_C(r)$ satisfies f.*

Example 10. Consider the following preference relation $\succ_{C_\alpha}$ where α is a selection condition over the schema *R:*

$$t_1 \succ_{C_\alpha} t_2 \equiv \alpha(t_1) \wedge \neg\alpha(t_2).$$

This is a very common preference relation expressing the preference for the tuples satisfying some property over those that do not satisfy it. The corresponding indifference relation $\sim_{C_\alpha}$ is defined as follows:

$$t_1 \sim_{C_\alpha} t_2 \equiv \alpha(t_1) \wedge \alpha(t_2) \vee \neg\alpha(t_1) \wedge \neg\alpha(t_2).$$

Theorem 7 implies that ω_{C_α} is redundant w.r.t. a set of CGDs F iff F implies the CGD

$$\forall t_1.\forall t_2.\ R(t_1) \wedge R(t_2) \Rightarrow \alpha(t_1) \wedge \alpha(t_2) \vee \neg\alpha(t_1) \wedge \neg\alpha(t_2).$$

The latter dependency is satisfied by an instance r of R if and only if all the tuples in r satisfy α or none does. In both cases $\omega_{C_\alpha}(r) = r$.

The paper [2] contains an effective reduction using *symmetrization* from entailment of CGDs to validity of $\forall$-formulas in the underlying constraint theory. (A similar construction using *symbol mappings* is presented in [29].) This immediately gives the decidability of the problems discussed in the present paper for equality and rational-order constraints (as well as other constraint theories for which satisfiability of quantifier-free formulas is decidable). A more detailed complexity analysis can be carried out along the lines of Theorems 2, 4, and 6.

For theorems 7, 8 and 9 to hold for a class of integrity constraints, two conditions need to be satisfied: (a) the class should be able to express constraints equivalent to d_2^C and d_3^C, and (b) the notions of entailment and finite entailment (entailment on finite relations) for the class should coincide. If (b) is not satisfied, then the theorems will still hold if reformulated by replacing "entailment" with "finite entailment". Thus, assuming that (a) is satisfied, the effectiveness of checking the preconditions of the above theorems depends on the decidability of finite entailment for the given class of integrity constraints.

7 Related Work

The basic reference for semantic query optimization is [6]. The most common techniques are: join elimination/introduction, predicate elimination and introduction, and detecting an empty answer set. [5] discusses the implementation of predicate introduction and join elimination in an industrial query optimizer. Semantic query optimization techniques for relational queries are studied in [29] in the context of denial and referential constraints, and in [25] in the context of constraint tuple-generating dependencies (a generalization of CGDs and classical relational dependencies). FDs are used for reasoning about sort orders in [27].

Two different approaches to preference queries have been pursued in the literature: qualitative and quantitative. In the *qualitative* approach, represented by [24, 14, 19, 3, 11, 7, 8, 17, 15, 18], the preferences between tuples in the answer to a query are specified directly, typically using binary *preference relations.* In the *quantitative* approach, as represented by [1, 13], preferences are specified indirectly using *scoring functions* that associate a numeric score with every tuple of the query answer. Then a tuple t_1 is preferred to a tuple t_2 iff the score of t_1 is higher than the score of t_2. The qualitative approach is strictly more general than the quantitative one, since one can define preference relations in terms of scoring functions However, not every intuitively plausible preference relation can be captured by scoring functions.

Example 11. There is no scoring function that captures the preference relation described in Example 1. Since there is no preference defined between any of the first three tuples and the fourth one, the score of the fourth tuple should be equal to all of the scores of the first three tuples. But this implies that the scores of the first three tuples are the same, which is not possible since the second tuple is preferred to the first one which in turn is preferred to the third one.

This lack of expressiveness of the quantitative approach is well known in utility theory [10, 9]. The importance of weak orders in this context comes from the fact that only weak orders can be represented using real-valued scoring functions (and for countable domains this is also a sufficient condition for the existence of such a representation [9]). In the present paper we do not assume that preference relations are weak orders. We only characterize a condition under which preference relations become weak orders relative to a set of integrity constraints.

[8,15,16] discuss algebraic optimization of preference queries.

8 Conclusions and Further Work

We have presented some techniques for semantic optimization of preference queries, focusing on the winnow operator. The simplicity of our results attests to the power of logical formulation of preference relations. However, our results are applicable not only to the original logical framework of [7,8], but also to preference queries defined using preference constructors [17,18] and skyline queries [3, 4, 23, 26] because those queries can be expressed using preference formulas.

Further work can address, for example, the following issues:

- identifying other semantic optimization techniques for preference queries,
- expanding the class of integrity constraints by considering, for example, tuple-generating dependencies and referential integrity constraints,
- identifying weaker but easier to check sufficient conditions for the application of our techniques,
- considering other preference-related operators like *ranking* [8].

References

1. R. Agrawal and E. L. Wimmers. A Framework for Expressing and Combining Preferences. In *ACM SIGMOD International Conference on Management of Data,* pages 297–306, 2000.
2. M. Baudinet, J. Chomicki, and P. Wolper. Constraint-Generating Dependencies. *Journal of Computer and System Sciences,* 59:94–115, 1999. Preliminary version in ICDT'95.
3. S. Börzsönyi, D. Kossmann, and K. Stocker. The Skyline Operator. In *IEEE International Conference on Data Engineering (ICDE),* pages 421–430, 2001.
4. J. Chomicki, P. Godfrey, J. Gryz, and D. Liang. Skyline with Presorting. In *IEEE International Conference on Data Engineering (ICDE),* 2003. Poster.
5. Q. Cheng, J. Gryz, F. Koo, C. Leung, L. Liu, X. Qian, and B. Schiefer. Implementation of Two Semantic Query Optimization Techniques in DB2 Universal Database. In *International Conference on Very Large Data Bases (VLDB),* 1999.
6. U. S. Chakravarthy, J. Grant, and J. Minker. Logic-Based Approach to Semantic Query Optimization. *ACM Transactions on Database Systems,* 15(2):162–207, 1990.
7. J. Chomicki. Querying with Intrinsic Preferences. In *International Conference on Extending Database Technology (EDBT),* pages 34–51. Springer-Verlag, LNCS 2287, 2002.
8. J. Chomicki. Preference Formulas in Relational Queries. *ACM Transactions on Database Systems,* 28(4):427–466, December 2003.
9. P. C. Fishburn. *Utility Theory for Decision Making.* Wiley & Sons, 1970.
10. P. C. Fishburn. Preference Structures and their Numerical Representations. *Theoretical Computer Science,* 217:359–383, 1999.
11. K. Govindarajan, B. Jayaraman, and S. Mantha. Preference Queries in Deductive Databases. *New Generation Computing,* pages 57–86, 2001.
12. S. Guo, W. Sun, and M.A. Weiss. Solving Satisfiability and Implication Problems in Database Systems. *ACM Transactions on Database Systems,* 21(2):270–293, 1996.
13. V. Hristidis, N. Koudas, and Y. Papakonstantinou. PREFER: A System for the Efficient Execution of Multiparametric Ranked Queries. In *ACM SIGMOD International Conference on Management of Data,* pages 259–270, 2001.
14. W. Kießling and U. Güntzer. Database Reasoning – A Deductive Framework for Solving Large and Complex Problems by means of Subsumption. In *3rd. Workshop On Information Systems and Artificial Intelligence,* pages 118–138. Springer-Verlag, LNCS 777, 1994.
15. W. Kießling and B. Hafenrichter. Optimizing Preference Queries for Personalized Web Services. In *IASTED International Conference on Communications, Internet and Information Technology,* November 2002. Also Tech. Rep. 2002-12, July 2002, Institute of Computer Science, University of Augsburg, Germany.

16. W. Kießling and B. Hafenrichter. Algebraic Optimization of Relational Preference Queries. Technical Report 2003-1, Institut für Informatik, Universität Augsburg, 2003.
17. W. Kießling. Foundations of Preferences in Database Systems. In *International Conference on Very Large Data Bases (VLDB),* 2002.
18. W. Kießling and G. Köstler. Preference SQL - Design, Implementation, Experience. In *International Conference on Very Large Data Bases (VLDB),* 2002.
19. G. Köstler, W. Kießling, H. Thöne, and U. Güntzer. Fixpoint Iteration with Subsumption in Deductive Databases. *Journal of Intelligent Information Systems,* 4:123–148, 1995.
20. G. Kuper, L. Libkin, and J. Paredaens, editors. *Constraint Databases.* Springer-Verlag, 2000.
21. A. Klug. Calculating Constraints on Relational Tableaux. *ACM Transactions on Database Systems,* 5:260–290, 1980.
22. A. Klug and R. Price. Determining View Dependencies Using Tableaux. *ACM Transactions on Database Systems,* 7, 1982.
23. D. Kossmann, F. Ramsak, and S. Rost. Shooting Stars in the Sky: An Online Algorithm for Skyline Queries. In *International Conference on Very Large Data Bases (VLDB),* 2002.
24. M. Lacroix and P. Lavency. Preferences: Putting More Knowledge Into Queries. In *International Conference on Very Large Data Bases (VLDB),* pages 217–225, 1987.
25. M. Maher and J. Wang. Optimizing Queries in Extended Relational Databases. In *International Conference on Database and Expert Systems Applications (DEXA),* pages 386–396, 2000.
26. D. Papadias, Y. Tao, G. Fu, and B. Seeger:. An Optimal and Progressive Algorithm for Skyline Queries. In *ACM SIGMOD International Conference on Management of Data,* pages 467–478, 2003.
27. David E. Simmen, Eugene J. Shekita, and Timothy Malkemus. Fundamental Techniques for Order Optimization. In *ACM SIGMOD International Conference on Management of Data,* pages 57–67, 1996.
28. R. Torlone and P. Ciaccia. Which Are My Preferred Items? In *Workshop on Recommendation and Personalization in E-Commerce,* May 2002.
29. X. Zhang and Z. M. Ozsoyoglu. Implication and Referential Constraints: A New Formal Reasoning. *IEEE Transactions on Knowledge and Data Engineering,* 9(6):894–910, 1997.

Constraint Processing Techniques for Improving Join Computation: A Proof of Concept

Anagh Lal and Berthe Y. Choueiry

Constraint Systems Laboratory
Department of Computer Science and Engineering
University of Nebraska-Lincoln
{alal,choueiry}@cse.unl.edu

Abstract. Constraint Processing and Database techniques overlap significantly. We discuss here the application of a constraint satisfaction technique, called dynamic bundling, to databases. We model the join query computation as a Constraint Satisfaction Problem (CSP) and solve it by search using dynamic bundling. First, we introduce a sort-based technique for computing dynamic bundling. Then, we describe the join algorithm that produces nested tuples. The resulting process yields a compact solution space and savings of memory, disk-space, and/or network bandwidth. We realize further savings by using bundling to reduce the number of join-condition checks. We place our bundling technique in the framework of the Progressive Merge Join (PMJ) [1] and use the XXL library [2] for implementing and testing our algorithm. PMJ assists in effective query-result-size prediction by producing early results. Our algorithm reinforces this feature of PMJ by producing the tuples as multiple solutions and is thus useful for improving size estimation.

1 Introduction

Although not widely acknowledged, progress made in the area of Databases has historically greatly benefited the area of Constraint Processing, and vice versa. We present here one new such opportunity, in which a Constraint Satisfaction technique that we have developed and call bundling is used to improve the computation of a join. The join operation is extensively studied in the database literature and remains one of the most computationally expensive operations. To the best of our knowledge, no work has yet exploited the existence of symmetries within a relation to improve the performance of the task or reduce the space necessary for storing the result. Consider the two relations `R1` and `R2` shown in Figure 1 (left), and the following SQL query:

```
SELECT R1.A, R1.B, R1.C
FROM R1, R2
WHERE R1.A = R2.A AND R1.B = R2.B  AND R1.C = R2.C
```

The natural join `R1` $\bowtie$ `R2` has the tuples shown in Figure 1 (center). Consider $\sigma_{A=2}(R1)$ and $\sigma_{A=4}(R1)$. The tuples they yield in `R1` $\bowtie$ `R2` differ only for the

B. Kuijpers and P. Revesz (Eds.): CDB 2004, LNCS 3074, pp. 143–160, 2004.

R1

A	B	C
1	12	23
1	13	23
1	14	23
2	10	25
3	16	30
3	16	24
4	10	25
5	12	23
5	13	23
5	14	23
6	13	27
6	14	27
7	14	28
7	19	20

R2

A	B	C
1	12	23
1	13	23
1	14	23
1	15	23
2	10	25
3	17	20
3	18	22
4	10	25
5	12	23
5	13	23
5	14	23
5	15	23
6	13	27
6	14	27
8	14	28

R1 ⋈ R2

A	B	C
1	12	23
1	13	23
1	14	23
2	10	25
4	10	25
5	12	23
5	13	23
5	14	23
6	13	27
6	14	27

R1 ⋈ R2 compacted

A	B	C
{1, 5} {2, 4} {6}	{12, 13, 14} {10} {13, 14}	{23} {25} {27}

Fig. 1. *Left:* R1, R2. *Center:* R1 ⋈ R2. *Right:* Compacted R1 ⋈ R2.

value of A. We say that A=2 and A=4 are symmetric and use this symmetry to compact the resulting tuples in R1 ⋈ R2. We propose here a technique for detecting such symmetries and exploiting them for compacting join results. Note that our technique is efficient but does not guarantee the maximal possible compaction. Our technique generates the compacted join relation shown in Figure 1 (right), which has only 3 tuples with nested values. When computing a sequence of join operations, the intermediate join results occupy less space. When these results are used in a subsequent join, more tuples are available per page, which reduces the I/O operations and thus saves time.

Although most research in Constraint Satisfaction focuses on binary constraints, many real-life problems are more 'naturally' modeled as non-binary Constraint Satisfaction Problems (CPSs). The focus on binary CSPs has so far been tolerated because it is always possible, in principle, to reduce a finite non-binary CSP to a binary one [3,4]. However, recent research has shown that it is sometimes more effective to operate on the non-binary encoding of the CSP than on its binary reduction [5]. At this point in time, research on non-binary constraints is still in its infancy. In [6,7], we proposed a technique, *dynamic bundling,* for dynamically detecting and exploiting symmetries in binary CSPs. In [8], we extended this technique to non-binary constraints and showed a significant improvement in processing time and solution space. In this paper, we show how to use dynamic bundling for processing join queries and compacting the join results. We make the following contributions:

1. Provide a new way to map a join query into a CSP (Section 3).
2. Present an algorithm for dynamic bundling that improves the memory usage of our previous implementation [8] and is more suitable for databases (Section 4).

3. Present a join algorithm for producing bundled solutions that are more compact, thus saving memory (Section 5).
4. Identify new opportunities for exploiting the compact solution space in other database applications such as data analysis and materialized views.

This paper is structured as follows. Section 2 states our motivation and provides background information. Section 3 models the join operator as a CSP. Section 4 describes a technique for bundling the values of an attribute in a relation. Section 5 uses this bundling technique in a new join algorithm. Section 6 discusses our implementation and the results of our experiments. Section 7 summarizes related work. Finally, Section 8 concludes the paper and gives directions for future research.

2 Background

In this section we explain the motivations behind our research and summarize background information useful for reading the paper.

2.1 Motivation

Join algorithms can be classified into three categories: hash-based, sort-based, and nested-loop algorithms. All these algorithms attempt to optimize the join by minimizing the number of times relations are scanned. Hash-based algorithms use hash-tables to partition relations according to the values of an attribute, and then join the partitions corresponding to the same values. The sort-based approach partitions relations by sorting them on the attributes involved in the join condition. Thanks to sorting, each tuple in a relation is compared with tuples of the other relation lying within a fixed range of values, which are significantly fewer than all possible tuples. Sorting reduces the number of scans of both relations and speeds up join processing. Nested-loop algorithms are used when relations fit in memory or when no adequate hashing function or useful sorting order is available. None of these techniques attempts to compact query results, although this can be beneficial given the large size of join results. The reduction of the number of I/O operations during query evaluation is a key factor in determining the efficiency of a database. Extensive research is devoted to the development of query-evaluation plans and evaluation algorithms that minimize the number of I/O operations. Our technique of dynamic bundling produces results that are compact by automatically detecting symmetries within a relation. Our goal is to exploit the use of these compact solution spaces in order to reduce I/O operations and extract information from query results useful for data analysis and data mining. We achieve this goal by first reducing the space requirements of our bundling technique in order to adapt it to the bundling of solution tuples of a query in the context of databases (Section 4). Then, we design a join algorithm that uses bundling (Section 5).

We project two other important uses of our technique, namely: improving query-size estimation and supporting data analysis and mining. Indeed, the fact

that the size of the compacted tuples produced by our technique is large is an indicator of high redundancy in the join relations. This information can be used to boost the estimate of query-result size, which is important for query planning. Further, the compacted results of our new join algorithm represent similar solutions that are clustered together. Let us consider a scenario with the two relations

```
Customer_Choice(Custid, Favorite_Product, Cust_Category),
```

which stores customer choices from an online survey, and

```
Customer_List(Custid),
```

which stores the customers staying in Lincoln, and a query to find the result of the online survey for Lincoln:

```
SELECT Customer_List.Custid, Favorite_Product, Cust_Category
FROM Customer_Choice, Customer_List
WHERE  Customer_Choice.Custid = Cust_List.Custid
```

Our techniques will produce results where customers with same product and category choices are bundled up together. This is just one example of how bundling adds information to query results. This additional information can be used for data mining and in packages for data analysis.

2.2 Sort-Based Join Algorithms

The join operator in relational algebra takes two relations as arguments and a condition (known as the join condition) that compares any two attributes, one from each of the two argument relations. The generic form of a join is $\mathsf{R} \bowtie_{x\theta y} \mathsf{S}$, where R and S are two relations, x and y are attributes from R and S respectively, and θ stands for a comparison operator (e.g., $=, \geq, \leq$, and $\neq$) called the *join condition.* Equality is the most commonly used join condition, and gives the *equi-join.* A *natural join* is a special case of an equi-join for which $x = y$, i.e. the attributes of the two relations are same. The join operation is among the most I/O-intensive operators of relational algebra because it may require multiple scans over the two input relations and also because the size of the result can be as large as the product of the sizes of these relations.

Our new join algorithm (Section 5) adopts the main idea of the Progressive Merge Join (PMJ) of [1]. PMJ is a join algorithm that produces query results early, and hence has the ability to provide valuable information to the query-size estimator. These are exactly the working conditions that we are targeting. PMJ is a special sort-merge join algorithm, which have two phases: the sorting phase and the merging phase. We first describe sort-merge algorithms in general, then discuss PMJ.

In the sorting phase of a sort-merge algorithm for computing the join of two relations, `R1` and `R2`, the memory of size M pages is first filled with pages of `R1`. These loaded pages are then sorted on the join-condition attributes and stored

back to disk as a sub-list or *run* of the relation. When R1 has N pages, $\frac{N}{M}$ runs are generated. This process is repeated for the second same-sized relation R2. At the end of the sorting phase, we have produced sorted runs of R1 and R2. Now, the merging phase can start. We first consider that $M \geq 2 \times \frac{N}{M}$. Now $\frac{M^2}{2 \times N}$ pages from each of the $\frac{N}{M}$ runs of R1 are loaded into memory, and the same is done for R2. The smallest unprocessed tuples from the pages of R1 and R2, respectively, are tested for the join condition. Those that satisfy the condition are joined and the result written as output. A page is exhausted when all its tuples have been processed. When a page is exhausted a page from the same run is brought in. When $M < 2 \times \frac{N}{M}$, multiple merge phases are needed. Each intermediate merging phase produces longer but fewer sorted runs. This process of generating longer but fewer runs continues until the number of runs of the two relations is equal to the number of pages that can fit in memory. This sort-merge algorithm is a *blocking* algorithm in the sense that the first results come only after the sorting phase is completed.

PMJ delivers results early by joining relations already during the sorting phase [1]. Indeed, during the sorting phase, pages from both the relations are read into memory, sorted, and joined to produce early results. Because PMJ produces results early, it is a *non-blocking* or pipelined version of the sort-merge join algorithm. The number of runs generated for each relation is more than that by a general sort-merge algorithm and is given by $\frac{M^2}{4 \times N}$. The merging phase is similar to that of a sort-merge algorithm, except that PMJ ensures that pages of R1 and R2 from the same run are not joined again as they have already produced their results in the sorting phase. The memory requirements of PMJ are more than those of a sort-merge algorithm because the number of runs generated during the sorting phase is double that of a sort-merge algorithm. The number of runs generated doubles because the memory is now shared by both relations. Because of the increased number of runs, the chances of multiple merging phases taking place increases. The production of early results causes the results of PMJ to be unsorted. However, the unsorted results allow for more accurate estimation of the result size, which is an important feature.

2.3 Constraint Satisfaction Problems

A Constraint Satisfaction Problem (CSP) is defined by $\mathcal{P} = (\mathcal{V}, \mathcal{D}, \mathcal{C})$ where $\mathcal{V} = \{V_i\}$ is a set of variables, $\mathcal{D} = \{D_{V_i}\}$ the set of their respective domains, and $\mathcal{C}$ a set of constraints that restrict the acceptable combination of values for variables. A constraint over the variables $V_i, V_j, \ldots, V_k$ is specified as $C_{V_i,V_j,\ldots,V_k} = \{(\langle V_i\ a_i\rangle, \langle V_j\ a_j\rangle, \ldots, \langle V_k\ a_k\rangle)\}$ with $a_i \in D_{V_i}, a_j \in D_{V_j}, \ldots, a_k \in D_{V_k}$. Solving a CSP requires assigning a value to each variable such that all constraints are simultaneously satisfied. The problem is **NP**-complete in general. The *scope* of a constraint is the set of variables to which the constraint applies, and its *arity* is the size of this set. Non-binary constraints are represented as hyper-edges in the constraint network. For sake of clarity, we represent a hyper-edge as another type of node connected to the variables in the scope of the constraint, see Figure 2.

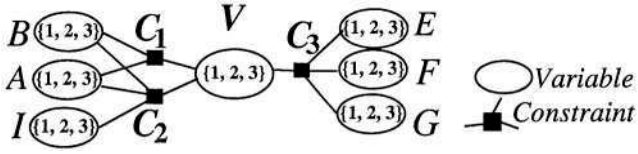

Fig. 2. Example of a non-binary CSP.

Solving CSPs with Search. CSPs are typically solved using depth-first search with backtracking. Depth-first search proceeds by choosing a current variable V_c and *instantiating* it, i.e. assigning to it a value a taken from its domain, $V_c \leftarrow a$. The variable and its assigned value define a variable-value pair (vvp) denoted by $\langle V_c\ a \rangle$. Uninstantiated variables are called future variables, and their set is denoted by $\mathcal{V}_f$. A look-ahead strategy called forward checking (FC) is then applied which removes from the domains of the future variables the values that are not consistent with the current assignment, thus *propagating* the effect of the instantiation $\langle V_c\ a \rangle$. V_c is then added to the set of instantiated variables, which we call past variables and denote as $\mathcal{V}_p$. If the instantiation does not wipe out the domain of any variable in $\mathcal{V}_f$, search considers the next variable in the ordering and moves one level down in the search tree. Otherwise, the instantiation is revoked, its effects are undone, and an alternative instantiation to the current variable is attempted. When all alternatives fail, search backtracks to the previous level in the tree. The process repeats until one or all solutions are found. Variables are considered in sequence according to a *variable ordering* heuristic. Common wisdom requires that the most-constrained variable be considered first in order to reduce the branching factor of the search tree and the number of backtracks.

2.4 Symmetry as Value Interchangeability

Interchangeability is a general concept that characterizes the types of symmetries that may arise in a CSP. The concept deals with redundancy in a CSP. In the broadest sense, when a CSP has more than one solution, one can define a mapping between the solutions such that one solution can be obtained from another without performing search. This is *functional* interchangeability [9]. We address here a restricted form of interchangeability: the interchangeability of values in the domain of a single variable. This type of interchangeability does not cover the permutation of values across variables, which is an isomorphic interchangeability. Consider the CSP shown in Figure 3 (A). In the absence of any symmetry consideration, the depth-first search process described in Section 2.3 yields the tree shown in Figure 3 (B). A simple analysis of the values of the variable V_2 shows that the values e and f are consistent with exactly the same values in the neighborhood of V_2, and consequently they are interchangeable in a solution of the CSP. Values e and f are said to be neighborhood interchangeable. Detecting neighborhood interchangeability can be efficiently done using the discrimination tree algorithm proposed in [9].

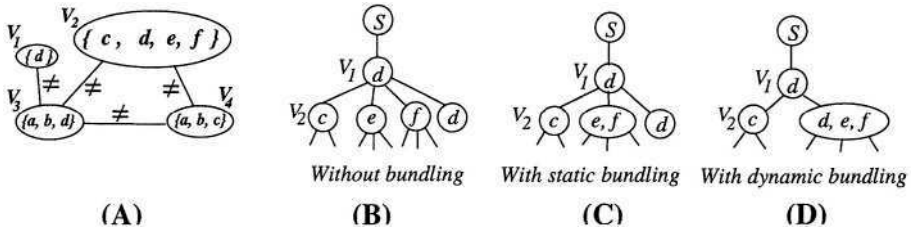

Fig. 3. Solving a CSP with search, with and without bundling.

Search with Static Bundling. Haselböck proposed to 'bundle' up, in the search tree, neighborhood interchangeable values (e.g., e and f for V_2) since they necessarily yield equivalent sub-trees [10], see Figure 3 (C). We call this technique *static* bundling because the bundles are computed *prior* to search.

Search with Dynamic Bundling. When using a look-ahead strategy such as forward checking for searching the CSP, the effect of an instantiation of a current variable is propagated to the domains of the future variables. In the example of Figure 3 (A), $V_1 \leftarrow d$ results in the elimination of d from the domain of V_3. At this point, one notices that all three values d, e, and f become neighborhood interchangeable for V_2. Dynamic bundling is based on the idea of recomputing the bundles as search proceeds to take advantage of the new opportunities to bundle values enabled by decisions taken along a path of the tree. Figure 3 (D) shows the tree generated by dynamic bundling. In previous work we have established that dynamic bundling is always beneficial: it yields larger bundles and reduces the search effort [6,7]. This unexpected result can be explained by the fact that, in addition to bundling solutions, dynamic bundling allows us to factor out larger no-goods (non solutions), thus eliminating more 'barren' portions of the search tree. Further, we showed that, in comparison to dynamic bundling, static bundling is prohibitively expensive, particularly ineffective, and should be avoided [7]. In [8], we extended this technique to non-binary constraints, and demonstrated significant improvements in processing time and solution space. In this paper, we show how to use dynamic bundling for processing join queries and compacting the join results.

3 Modeling a Join Query as a CSP

We show how to model a join query as a CSP using our running example:

```
SELECT R1.A, R1.B, R1.C
FROM R1, R2
WHERE R1.A = R2.A AND R1.B = R2.B AND R1.C = R2.C
```

We map the join query into the following CSP $\mathcal{P} = \{\mathcal{V}, \mathcal{D}, \mathcal{C}\}$, represented by the constraint network of Figure 4:

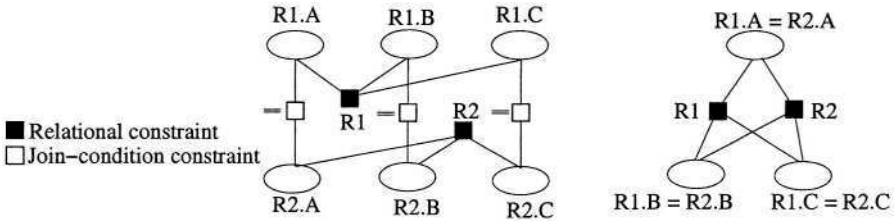

Fig. 4. Two equivalent formulations of the join as a CSP.

1. *The attributes as CSP variables.* $\mathcal{V}$ is the set of attributes in the join query. There are 6 variables in our example, which are the attributes `R1.A`, `R2.A`, `R1.B`, `R2.B`, `R1.C`, and `R2.C`.
   For an equi-join query, as it is the case here, the attributes joined using an equality constraint can be represented by a unique variable. In the example above, the CSP representing the query would consist of only 3 variables with `R1.A=R2.A`, `R1.B=R2.B`, and `R1.C=R2.C`. When the query lists the two equated attributes in its `SELECT` clause, the CSP variable is simply repeated in the output.
2. *The attribute values as variable domains.* $\mathcal{D}$ is the set of the domains of the variables. For each attribute, it is the set of values the attribute takes in a relation.
   For an equi-join query, the domain of the CSP variable representing the equated attribute is the union of the set of values that the equated attributes take in their respective relations.
3. *The relations and join conditions as CSP constraints.* $\mathcal{C}$ is the set of constraints of the CSP. These constraints originate from two sources, namely: the relations to be joined and the join conditions. The relations to be joined directly map to CSP constraints that are expressed extensionally. We call these constraints *relational constraint.* The join conditions map directly to CSP constraints expressed intensionally, which we call *join-condition constraints.* In our example, the relations to be joined are `R1` and `R2`, and there are 3 equality constraints due to the join conditions of the query.
   For an equi-join query where the equated attributes are represented by a unique variable, the join condition is implicit in the CSP representation and does not need to be expressed.

Table 1 maps the terminology of databases into that of Constraint Processing.

Our algorithm for bundling non-binary CSPs required that constraints be enumerated [8]. However, for computing interchangeability in the database scenario, we do not have to enumerate the join-condition constraints and store them explicitly. Instead, we proceed as follows. When joining two relations specified in extension, the resulting tuple is checked for consistency with the join-conditions specified in intension as this tuple is being built up. When the values in the partially built tuple are not consistent with a join-condition constraint, the tuple is discarded, as we explain in Section 5.1. This is possible because we are guaranteed that all the CSP variables are present in at least one constraint de-

Table 1. Terminology mapping.

DB terminology	CSP terminology
Table, relation	Constraint (which we call relational constraint)
Join condition	Constraint (which we call join-condition constraint)
Relation arity	Constraint arity
Attribute	CSP variable
Value of an attribute	Value of a CSP variable
Domain of an attribute	Domain of a CSP variable
Tuple in a table	Tuple in a constraint Tuple allowed by a constraint Tuple consistent with a constraint
Constraint relation (in Constraint Databases)	Constraint of linear (in)quality
A sequence of natural joins	All the solutions of the corresponding CSP

fined in extension and thus all the join-condition constraints will be checked for consistency.

4 Bundling Relations

This section describes the computation of interchangeable values (i.e., a bundle) of an attribute in a relation. Since our join algorithm is a sort-merge algorithm, the relations must first be sorted. Thus, we need to select the order of the attributes for sorting the relations. This order is necessarily static because we cannot afford to re-sort relations during processing. In terms of CSPs, this corresponds to a static ordering of the variables. We first describe our ordering heuristic then the technique for computing interchangeability.

4.1 Heuristic for Variable Ordering

With $\mathcal{V}$ the set of variables in the CSP representing a query, we denote $\mathcal{V}_q$ a first-in first-out queue of the ordered variables. $\mathcal{V}_q$ is initialized to an arbitrary variable[1]. We also denote $\mathcal{V}_u$ the unordered variables (i.e., $\mathcal{V}_u = \mathcal{V} \setminus \mathcal{V}_q$). Let V_c be the last variable added to $\mathcal{V}_q$. The next variable in the order V_n is chosen from $\mathcal{V}_u$ as follows:

1. Consider the variables $\{V_i\} \subseteq \mathcal{V}_u$ such that V_i is linked with a join-condition constraint C_i to V_c. V_n is selected as the variable for which $| \mathcal{V}_u \cap scope(C_i) |$ is the smallest.
2. V_n is selected as a variable from the same relation as V_c.
3. V_n is selected arbitrarily from $\mathcal{V}_u$.

[1] One can elaborate heuristics for choosing the first variable. One possibility is to exploit the meta-data maintained by the DBMS such as the number of unique values of an attribute. Other heuristics may choose first the attribute that participates in the largest number of constraints. The design and evaluation of such heuristics still needs to be investigated.

If no variables satisfy a rule in the sequence above, the next rule in sequence is applied to $\mathcal{V}_u$. When more than one variable satisfy a rule, the next rule in sequence is applied to discriminate among the qualifying variables. V_n is removed from $\mathcal{V}_u$ and added to $\mathcal{V}_q$. The process is repeated until $\mathcal{V}_u$ is empty. The goal of this ordering is to allow the checking of join-condition constraints as early as possible. For the example of Figure 4, one possible ordering is the sequence `R1.A, R2.A, R1.B, R2.B, R1.C,` and `R2.C`. Note that the ordering of the variables affects the size of the generated bundles and that different ordering heuristics need to be investigated.

4.2 The Principle

Given the queue $\mathcal{V}_q$ of ordered variables, we build the bundles dynamically while joining the tuples loaded in memory. Variables in the queue are considered in sequence. The variable under consideration is called the current variable V_c, the set of previous ones is called $\mathcal{V}_p$, and the set of remaining ones $\mathcal{V}_f$. $\mathcal{V}_f$ is initialized to $\mathcal{V}_q$, keeping the same order of variables, and $\mathcal{V}_p$ is set to nil. First, we find a bundle for V_c as described below. Then, we determine the subset of values in the bundle that is consistent with at least one bundle from each of the variables in $\mathcal{V}_f$ with a join-condition constraint with V_c (see Algorithm 2). If such a subset is not empty, we assign it to V_c. In terms of CSPs, this corresponds to instantiating V_c. We move V_c to $\mathcal{V}_p$, and a new V_c is chosen as the first variable in $\mathcal{V}_f$. Otherwise, if the consistent subset for V_c is empty, we compute the next bundle of V_c from the remaining tuples and repeat the above operation. We continue this process until all the variables are instantiated and then output these instantiations as the next nested tuple of the join. Consider the scenario where a next bundle for V_c, an attribute of relation `R1`, needs to be computed during a sequence of instantiations (see Figure 5).

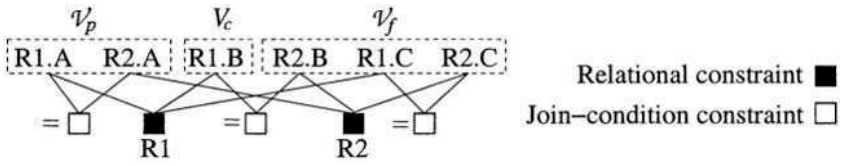

Fig. 5. Instantiation sequence.

The bundle depends on the instantiation of variables from `R1` in $\mathcal{V}_p$ (i.e., previously instantiated variables). Although the computed bundle of V_c does not directly depend on the instantiations of past variables from `R2`, the bundle subset to be assigned to V_c must be consistent with those variables of $\mathcal{V}_p$ that share a join-condition constraint with V_c. When such a variable is from `R2`, then the instantiation of V_c is affected by the instantiations of variables from `R2`.

Below, we describe the method for computing a bundle of V_c, an attribute of relation `R`, given that some of the variables of `R` are in $\mathcal{V}_p$. The bundles are

computed on the tuples of R present in the memory, called R′. First, R′ is sorted with the variable coming earliest in the static ordering (see Section 4.1) as the primary key, the one coming second as the secondary key, and so on. The sorting clusters tuples with the same values for variables as they appear in the static ordering.

4.3 Data Structures

We first introduce the various data structures used for computing the bundles.

- *Current-Inst* is a record of size equal to the number of variables in the CSP. It is used to store the current instantiations of variables of R in $\mathcal{V}_p$. This corresponds to a current path in a search tree. When a variable is assigned a bundle of size greater than one, only the smallest value in the bundle is stored in *Current-Inst,* as a representative of the bundle.
- *Processed- Values* is a similar record storing cumulatively all non-representative values of the assigned bundles. While computing bundles of V_c, tuples corresponding to values for V_c in *Processed-Values* are ignored.
- *Current-Constraint* is a selection of the relation R' (of which V_c is an attribute) such that: (1) Past variables have the values stored in *Current-Inst,* and (2) the value of V_c is greater than the previous instantiation of V_c. Initially, the *Current- Constraint* is set to R'.

The tuples with the same value for V_c in *Current-Constraint* form a partition p, and the value of V_c in this partition is denoted VALUE(p). Figure 6 shows these data-structures under various scenarios. A partition p is marked as *checked* when VALUE(p) is part of an instantiation bundle or when p is selected to be compared with other partitions. Otherwise, the partition is considered *unchecked.* P_c refers to the unchecked partition with the lowest value of V_c in *Current-Constraint.* When no checked partition exists for V_c, P_c is set to a dummy such as -1.

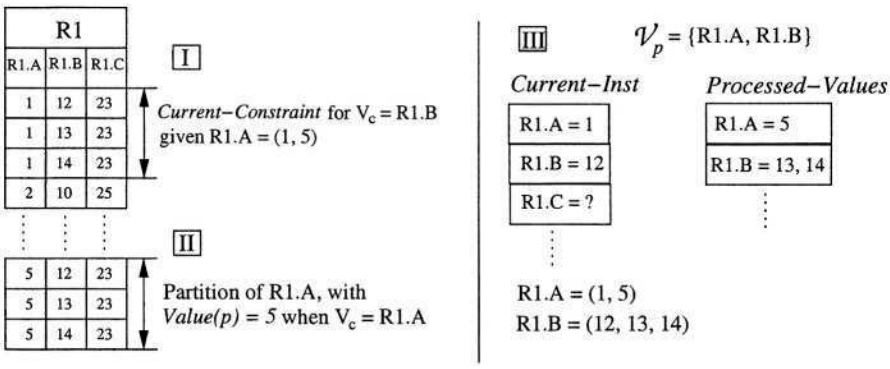

Fig. 6. Data structures shown under 3 different scenarios.

4.4 Bundle Computation

Algorithm 1 computes the next bundle of V_c given P_c. NEXT-PARTITION(p) returns the first *unchecked* partition in *Current-Constraint* following the partition p. For p= -1, NEXT-PARTITION(p) returns the first partition in *Current-Constraint*. P_c moves to the next unchecked partition at every call of Algorithm 1.

Input: V_c, *Current-Constraint*
bundle ← `nil`, the bundle to return
P_c ← NEXT-PARTITION(P_c)
Mark P_c as checked
Push VALUE(P_c) into *bundle*
P'_c ← NEXT-PARTITION(P_c)
while P'_c **do**
  t ← tuples of P_c
  p ← tuples of P'_c
  if $\pi_{v_f}(t) \equiv \pi_{v_f}(p)$ **then** push VALUE(P'_c) in *bundle*
  P'_c ← NEXT-PARTITION(P'_c)
end
Output: *bundle*

Algorithm 1: Algorithm to generate the next bundle of V_c.

Algorithm 1 is called by Algorithm 2 of Section 5 for computing the bundle b_c of V_c and the bundles of the variables V_i connected to V_c with a join-condition constraint. Further, Algorithm 2 determines the subset *Inst* of the bundle b_c that is consistent with the variables V_i. This consistent set of values *Inst* is then used to instantiate V_c. This instantiation operation includes the update of the data structures *Current-Inst* and *Processed-Values*. In particular, the values in *Processed-Values* that are lesser than those associated with P_c are deleted.

We can compute all the bundles of V_c by repeatedly calling Algorithm 1, then assigning the returned bundle to V_c until Algorithm 1 returns `nil`. Thus, the algorithm described here implements a lazy approach for computing the bundles and avoids storing the entire partition of the domain of every variable.

In the method described above *Processed-Values* is the data structure that occupies the most space. Whereas all the other data structures have sizes proportional to the number of variables (and therefore cause insignificant memory overhead), the size of *Processed-Values* depends on the number of tuples and the amount of bundling performed. The worst-case scenario for *Processed-Values* occurs when all the values of a variable are in a single bundle. In this case, *Processed- Values* will hold all the unique values of that variable. Suppose that there are N tuples in the relation, the relation has k attributes, and the number of unique values of the variable is $\frac{N}{l}$, where l is the average length of each partition of V_c. Then, the size of *Processed-Values* is $\frac{N}{l \times k}$ tuples. However, if this bundle goes on to form a result tuple, it will save more space than required for bundling. Even when this bundle fails to yield a result tuple, it still saves on

many comparisons thereby speeding up computation. Our current implementation is a proof of concept, and we are investigating how to improve its efficiency, possibly by the use of bit-maps.

5 Join Algorithm Using Bundling

This section shows the use of bundling while computing a join as a depth-first search. The join algorithm discussed in this section is based on the Progressive Merge Join. The technique discussed here can be easily adapted to the simpler sort-merge join since PMJ is just an extension of sort-merge. We first describe the in-memory join algorithm, and then place it in the schema of the external join algorithm.

5.1 Join Computation in Memory

We present here the algorithm to join two sub-sets of relations that are currently in memory. For the sake of readability, Algorithm 2 is restricted to binary join conditions (where the join conditions are between two attributes from different relations). It can be easily extended to join conditions with more than two attributes. Algorithm 2 takes as input the level of V_c in the search tree (i.e., *depth*) and the current path represented by the data structure *Current-Solution*. *Current-Solution* is a record that stores the assigned bundles to the variables in $\mathcal{V}_p$ (note that *Current-Solution* cannot be obtained from *Current-Inst* and *Processed-Values*). Variable[] is the array of variables in the same order as the static ordering of Section 4.1. When BACKTRACK is called the value for *Variable*[*depth*] in *Current-Inst* is reset, the *Processed-Values* for the variable is emptied, the value for the variable in *Current-Solution* is reset, and *Current-Constraint* is re-computed, thus undoing the effects of the previous instantiation. The function COMMON() computes the set of values in the input bundles that are consistent with each other according to the applicable join-condition constraints. Because this algorithm combines sorting and constraint propagation with bundling, it produces solutions quickly, which compensates for the effort spent on bundling.

5.2 The Structure of the Overall Join

We have discussed join computation of tuples that are in memory and now describe the steps for computing the join of complete relations using our in-memory join algorithm, Algorithm 2. The join of the two input relations is computed using an approach similar to the PMJ, in the two phases shown below.

Sorting Phase. The sorting phase is similar to the PMJ, except that for joining the pages of relations in memory we use the bundling-based technique of Algorithm 2. The sorting phase produces the early results and also a sorted sub-list or runs of the relations. These runs are stored back on disk and used in the

```
Input: depth, Current-Solution
while (depth ≤| V |) and (depth ≥ 1) do
    V_c ← Variable[depth]
    b_c ← next bundle for V_c using Algorithm 1
    if b_c is empty then
        BACKTRACK, decrement depth, and GOTO L1
    end
    Inst ← b_c
    repeat
        foreach V_i ∈ V_f connected to V_c by a join-condition constraint do
            Consider R_i the relational constraint that applies to V_i
            Select r_i from R_i according to Current-Solution
            repeat
                Find a bundle b_i applying Algorithm 1 on V_i and r_i
                if b_i is empty then break
                I_i ← COMMON(b_i, b_c)
            until I_i is not empty;
            if no b_i then BACKTRACK, decrement depth and Goto L1
        end
        Inst ← COMMON(I_0, I_1, ..., I_n)
    until Inst is not empty;
    Instantiate V_c with Inst
    Current-Solution[V_c] ← Inst
    Increment depth
    L1:
end
Output: Current-Solution
```

Algorithm 2: Algorithm to compute the in-memory join using bundling.

merging phase of the join. Since the memory is filled with pages from both the relations, the number of runs generated for each relation is $\frac{2N}{M}$.

Merging Phase. In the merging phase, as for the PMJ, $\frac{M^2}{4\times N}$ pages from every run created from the sorting phase are kept in memory. Let P_i^{rel} represent the pages in memory of relation *rel* and i^{th} run, where $rel \in \{0,1\}$ and $i \in \{1,2,\ldots,\frac{2N}{M}\}$. We store one solution each from the join of pages in an array, called *solution,* defined by Equation (1).

$$solution[i][j] = P_i^0 \bowtie P_j^1\ ,\ i \neq j \tag{1}$$

The minimum solution from solution[][]is the next result of the join. The next solution from the pages that gave the minimum solution is then computed and used to fill the corresponding place in *solution*[][]. A page P_i^{rel} is removed from memory and replaced with another page from the same run only if it satisfies the following two conditions for every page P_j^{1-rel}. P_i^{rel} is being joined with: (1) No

more join tuples result from $P_i^{rel} \bowtie P_j^{1-rel}$, and (2) the last tuple in P_i^{rel} is less than that of P_j^{rel}. The tuples are compared using the same comparison criteria as the ones used for sorting. These conditions ensure the tuples are produced in sorted order (during the merging phase) and that the algorithm is complete.

6 Implementation and Experiments

One of the goals of the XXL library [2] is to provide an infrastructure for evaluating new algorithms in databases. For example, PMJ was evaluated experimentally using this library. In our experience, XXL provides a good infrastructure for building new database algorithms through its rich cursor algebra built on top of Java's iterator interface. We implemented our join algorithm by extending the BUFFEREDCURSOR class of the XXL library.

The current implementation is a proof of concept and offers much room for improvement. To show the feasibility of our technique, we tested our join algorithm on randomly generated relations and on data from a real-world resource allocation problem in development in our group. For the real-world application, we computed the sequence of the natural join of three relations, with respectively 3, 4, and 3 attributes. The corresponding CSP has 4 variables, with domain size 3, 3, 300, and 250 respectively. The resulting join of size 69 was compressed down to 32 nested tuples. For the random problems, we used relations of $n = 10{,}000$ tuples. We set the page size to 200 tuples and the available memory size to $M = \frac{2N}{5}$, where N = 10000/200. We executed the query of our running example over five such pairs of relations. The result of the query had an average of 8,500 tuples, signifying that the query was selective. The number of tuples produced by bundling was reduced to 5,760 bundled tuples, an average of 1.48 tuples per bundle. The number of pages saved was more than $\frac{N}{4}$. Even if the worst-case scenario for the join occurred for every in-memory join (which is a highly unlikely event), the additional cost due to bundling is given by $\frac{N}{l \times k}$, where $\frac{N}{l}$ is the number of unique values of an attribute and k is the number of attributes in one relation (which is 3 here). For the worst case when $l = 1$, there are still savings in terms of pages. Again, the worst-case described here is of the current implementation, which offers much room for improvement.

7 Related Work

The idea of data compression is not new and is used in compressed database systems [11]. In these systems, data is stored in a compressed format on disk. It is decompressed either while loading it into memory or while processing a query. The compression algorithms are applied at the attribute level and are typically dictionary-based techniques, which are less CPU-intensive than other classical compression techniques [12]. Although most of the work in compressed databases applied to relations with numerical attributes [11], some work on string attributes has also been done [13]. Another feature of compressed databases that differs from our approach is that the query results passed to the next operator are

uncompressed and likely to be large. Our work differs from the above in that we reduce some of the redundancy present between tuples of a given relation. Our techniques are independent of the data type of an attribute. Further, the results of our queries are compacted, thereby assisting the next operator and reducing the storage of materialized views on disk. When these compacted results are loaded into memory for query processing, the de-compaction is effectively cost-free. The only costs associated with our techniques are those for performing the compaction. Finally, the compaction is carried out while the query is being evaluated, and is not a distinct function performed in separation.

In [14], Mamoulis and Papadias present a spatial-join algorithm using mechanisms of search with forward checking, which are fundamental in Constraint Processing. They store the relations representing spatial data in R-tree structures and use the structures to avoid unnecessary operations when computing a join. The constraints under consideration are binary. The key idea is to reduce the computational cost by propagating the effects of search, thereby detecting failure early. Our technique is not restricted to binary constraints, and is applicable to constraints of any arity. Further, it differs from the approach of Mamoulis and Papadias in that it reduces I/O operation and compacts join results in addition to reducing computational operations.

Bernstein and Chiu [15], Wallace et al. [16], Bayardo [17], Miranker et al. [18] exploit the standard consistency checking techniques of Constraint Processing to reduce the number of the intermediate tuples of a sequence of joins. While Wallace et al. consider Datalog queries, Bayardo and Miranker et al. study relational and object-oriented databases. Our CSP model of join query differs from their work in that the constraints in our model include both relational and join-condition constraints, whereas the latter models the relational constraints as CSP variables and only the join-condition constraints as CSP constraints. Thus, our model is finer in that it allows a more flexible ordering of the variables of the CSP, which increases the performance of bundling. Finally, Rich et al. [19] propose to group the tuples with the same value of the join attribute (redundant value). Their approach does not bundle up the values of the join attribute or exploit that redundancies that may be present in the grouped sub-relations.

8 Conclusions and Future Work

We described a new method for computing interchangeability and use it in a new join algorithm, thus establishing the usefulness of dynamic bundling techniques for join computation. In the future, we plan to address the following the issues:

- Refine our implementation by the use of lighter data structures.
- Test the usefulness of these techniques in the context of constraint databases where the value of an attribute is a continuous interval such as spatial databases [20].
- Conduct thorough evaluations of overall performance and overhead (memory and cpu) on different data distributions. And,
- Investigate the benefit of using bundling for query size estimation and materialized views.

Acknowledgments

This work is supported by the Maude Hammond Fling Faculty Research Fellowship and CAREER Award #0133568 from the National Science Foundation. We are grateful to the reviewers for their comments.

References

1. Dittrich, J.P., Seeger, B., Taylor, D.S., Widmayer, P.: On Producing Join Results Early. In: 22^{nd} ACM Symposium on Principles of Database Systems. (2003) 134–142
2. den Bercken, J.V., Blohsfeld, B., Dittrich, J.P., Krämer, J., Schäfer, T., Schneider, M., Seeger, B.: XXL–A Library Approach to Supporting Efficient Implementations of Advanced Database Queries. In: 27^{th} International Conference on Very Large Data Bases. (2001) 39–48
3. Rossi, F., Petrie, C., Dhar, V.: On the Equivalence of Constraint Satisfaction Problems. In: Proc. of the 9^{th} ECAI, Stockholm, Sweden (1990) 550–556
4. Bacchus, F., van Beek, P.: On the Conversion between Non-Binary and Binary Constraint Satisfaction Problems Using the Hidden Variable Method. In: Proc. of AAAI-98, Madison, Wisconsin (1998) 311–318
5. Bessière, C., Meseguer, P., Freuder, E.C., Larrosa, J.: On Forward Checking for Non-binary Constraint Satisfaction. Artificial Intelligence **141 (1-2)** (2002) 205–224
6. Beckwith, A.M., Choueiry, B.Y., Zou, H.: How the Level of Interchangeability Embedded in a Finite Constraint Satisfaction Problem Affects the Performance of Search. In: AI 2001: Advances in Artificial Intelligence, 14*th* Australian Joint Conference on Artificial Intelligence. LNAI Vol. 2256, Adelaide, Australia, Springer Verlag (2001) 50–61
7. Choueiry, B.Y., Davis, A.M.: Dynamic Bundling: Less Effort for More Solutions. In Koenig, S., Holte, R., eds.: 5th International Symposium on Abstraction, Reformulation and Approximation (SARA 2002). Volume 2371 of Lecture Notes in Artificial Intelligence., Springer Verlag (2002) 64–82
8. Lal, A., Choueiry, B.Y.: Dynamic Detection and Exploitation of Value Symmetries for Non-Binary Finite CSPs. In: Third International Workshop on Symmetry in Constraint Satisfaction Problems (SymCon'03), Kinsale, County Cork, Ireland (2003) 112–126
9. Freuder, E.C.: Eliminating Interchangeable Values in Constraint Satisfaction Problems. In: Proc. of AAAI-91, Anaheim, CA (1991) 227–233
10. Haselböck, A.: Exploiting Interchangeabilities in Constraint Satisfaction Problems. In: Proc. of the 13^{th} IJCAI, Chambéry, France (1993) 282–287
11. Roth, M.A., Horn, S.J.V.: Database compression. SIGMOD Record **22** (1993) 31–39
12. Westmann, T., Kossmann, D., Helmer, S., Moerkotte, G.: The implementation and performance of compressed databases. SIGMOD Record **29** (2000) 55–67
13. Chen, Z., Gehrke, J., Korn, F.: Query optimization in compressed database systems. In: 2001 ACM International Conference on Management of Data (SIGMOD). (2001) 271–282
14. Mamoulis, N., Papadias, D.: Constraint-based Algorithms for Computing Clique Intersection Joins. In: Sixth ACM International Symposium on Advances in Geographic Information Systems. (1998) 118–123

15. Bernstein, P.A., Chiu, D.M.W.: Using semi-joins to solve relational queries. J. ACM **28** (1981) 25–40
16. Wallace, M., Bressan, S., Provost, T.L.: Magic checking: Constraint checking for database query optimization. In: CDB 1995. (1995) 148–166
17. Bayardo, R.J.: Processing Multi-Join Queries. PhD thesis, University of Texas, Austin (1996)
18. Miranker, D.P., Bayardo, R.J., Samoladas, V.: Query evaluation as constraint search; an overview of early results. In Gaede, V., Brodsky, A., Günther, O., Srivastava, D., Vianu, V., Wallace, M., eds.: Second International Workshop on Constraint Database Systems (CDB '97). LNCS 1191, Springer (1997) 53–63
19. Rich, C., Rosenthal, A., Scholl, M.H.: Reducing duplicate work in relational join(s): A unified approach. In: International Conference on Information Systems and Management of Data. (1993) 87–102
20. Revesz, P.: Introduction to Constraint Databases. Springer-Verlag, New York (2001)

Taking Constraints out of Constraint Databases

Dina Q. Goldin*

University of Connecticut, USA
dqg@cse.uconn.edu

Abstract. By providing a finite representation for data with infinite semantics, the Constraint Database approach is particularly appropriate for querying spatiotemporal data. Since the introduction of CDBs in the early 1990's, several CDB prototypes have been implemented; however, CDBs have yet to prove their commercial viability.
In this paper, we rethink the main contribution behind the Constraint Database (CDB) framework; it's not about constraints! Rather, the main contribution of CDBs is in providing an *additional layer* to the database architecture, by splitting the *logical* into *abstract* and *concrete* layers. The abstract layer, representing an infinite relational extent of the spatiotemporal data, is the one over which CDB queries are written. The finite data representation is in the concrete layer; traditionally, constraints are used for this representation. By contrast, in current GIS and spatial databases, the representation is based on geometrical structure rather than constraints.
We propose a single three-tier architecture that combines these approaches. Its top abstract layer represents the infinite relational extent of the data, but the concrete layer admits both constraint-based and geometric data representations. The resulting *constraint-backed* systems, while preserving the advantages of CDBs, are more practical than *pure* constraint databases. By admitting more flexible CDB architectures, where data is not necessarily represented with constraints, significant progress can be made towards producing commercially successful CDB systems.

1 Introduction

By providing a finite representation for data with infinite semantics, the Constraint Database approach is particularly appropriate for querying spatiotemporal data. Since the introduction of CDBs in the early 1990's, several CDB prototypes have been implemented. However, CDBs have yet to prove their commercial viability.

In this paper, we rethink the main contribution behind the Constraint Database (CDB) framework; it's not about constraints! Rather, the main contribution of CDBs is to increase the level of abstraction between the physical data and its semantics, by introducing an *additional layer* to the database architecture. This is accomplished by splitting the *logical* into *abstract* and *concrete* layers. The abstract layer, representing an infinite relational extent of the spatiotemporal data,

* Supported by NSF grants no. 9733678 and IRI-0296195.

B. Kuijpers and P. Revesz (Eds.): CDB 2004, LNCS 3074, pp. 161–172, 2004.

is the one over which CDB queries are written. The finite data representation is in the concrete layer; traditionally, constraints are used for this representation.

By contrast, in current GIS and spatial databases, the abstract representation is based on *geometrical structure* rather than constraints. The geometry is restricted to two dimensions, and is supported by a large class of specialized operators, such as the following *topological operators* proposed by the *Open GIS Consortium* [15]: *Equal, Disjoint, Intersect, Touch, Cross, Within, Contains, Overlap, Relate.*

We propose a single three-tier architecture that combines these approaches. Its top abstract layer represents the infinite relational extent of the data, but the concrete layer admits both constraint-based and geometric data representations. The resulting *constraint-backed* systems, while preserving the advantages of CDBs, are more practical than *pure* constraint databases, where all data representation is in terms of constraints.

We believe that by admitting more flexible CDB architectures, where data is not necessarily represented with constraints, significant progress will be made towards producing commercially successful CDB systems.

Overview. In section 2, we show that the main contribution of constraint databases is in the providing an additional layer to the data model. In section 3, we explain why it is not necessary for the concrete layer of constraint databases to be based exclusively on constraints. To confirm our position, we present several case studies of spatiotemporal systems, both constraint-based and otherwise, in section 4. In section 5, we point out some disadvantages of the constraint representation for spatial data, arguing that geometric representation is sometimes preferable for spatiotemporal data. We summarize in section 6.

2 From Two Layers to Three

In this section, we compare relational and constraint data models, focusing on their *multi-tiered architecture.* The addition of a third tier in constraint databases is viewed as the major difference between them.

2.1 Relational Data Model

The relational data model, proposed by Codd in [6], can be considered the most significant achievement of database theory. While the pre-relational database models required queries to be aware of the physical data layout, the relational model separated this *physical* layer from the *logical* layer, and provided *relations* as a simple and uniform representation of data at the logical level. Queries were based on the logical level, freeing the user from dependence on physical data layout.

Logical layer. The logical layer is the layer over which user queries are specified. It defines the data representation proposed to the user, specifically the table-based representation with set-of-points semantics.

Physical layer. The physical layer describes the physical storage structure of the database. It manages the organization of files on disks and access paths.

The advantages of the relational approach were profound. Relational queries have clean and simple declarative semantics, and furthermore they need not be rewritten if the underlying database implementation is changed. Eventually, relational databases became a success beyond all original expectations, dominating all other database models both in academia and commercially by the late 1980s.

Their applicability of the relational model is limited to the sphere of *administrative* data, where the attributes range over discrete values such as birth dates, prices, or order numbers. However, this limitation was not seen as a concern as long as such data dominated all commercial data management applications. With proliferation of personal computing and multimedia applications, it eventually became clear that more and more of the data is not administrative, and therefore this limitation of relational databases became more pressing. Object-oriented technology offered a solution in the 1980's [2], of encapsulating non-administrative data and providing methods to process it within such encapsulation. Object-relational databases, which provide this object-oriented functionality within a mostly relational framework, are the latest commercial success, replacing relational databases.

2.2 Constraint Data Model

Constraint Databases (CDBs), proposed by Kanellakis et. al. in [13,14], represent a newer approach to addressing the problem of managing non-traditional data. While their contribution is usually described as "representing the data and the queries by constraints", this view does not get to the heart of the matter. We believe that the following is a better description of the CDB approach:

> Constraint databases allow relations that include *infinitely* many data points, by replacing the notion of *finite data* with *finitely representable* data.

To quote from [9], *finite relations are generalized to finitely representable relations.* In fact, the term *generalized* was used throughout [9] and other early works, both for tuples and relations; only later did *constraint* replace *generalized* as a preferred adjective.

Basically, the constraint data model replaces the logical layer of the relational model by two layers, *abstract* and *concrete* (see figure 1). The abstract level represents *data semantics,* and the concrete level represents the *logical structure of data.* By separating the (infinite) relational semantics of the data from its a finite representation, a database framework is obtained that can handle non-traditional data, while providing clean query semantics.

Note that the same two levels can be observed in relational databases, where data is represented abstractly as *sets of tuples,* and concretely as *tables of entries.* The one-to-one correspondence between the semantics and the logical structure of administrative data allowed the collapse of these levels into one in the case of

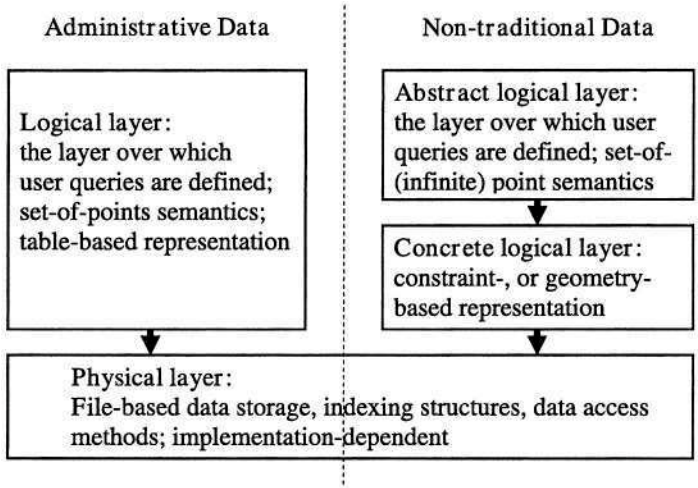

Fig. 1. Relational and Constraint Data Models: from 2 layers to 3.

the relational model. But in the case of non-traditional data, the correspondence is no longer trivial, hence the separation.

3 Representing the Concrete Layer

From the start, the main selling point of constraint databases has been the fact that their abstract layer provides a clean and declarative semantics for querying non-traditional data. Since then, many spatiotemporal database researchers, including those outside the CDB community, have discovered the advantages that come with providing an abstract layer based on infinite relational semantics. For example, this layer can be found in the data model for *interpolated data* [11], as well as in the data model for *moving objects* [12] – though neither work is directly related to constraint databases.

Another selling point of constraint databases has been their ability to model non-traditional data of arbitrary dimensionality, providing uniform query primitives for all geometric as well as alphanumeric data. This advantage was due to the use of constraints for *all* representation at the concrete level. However, constraints are not necessary to obtain this advantage. The choice of representation for the concrete layer is transparent to the user, since query formulation is based on the abstract layer. In principle, any representation that can be put into correspondence with the point-based semantics of the abstract layer, and implemented efficiently at the physical layer, can be used for any given relation.

The advantages of the three-level approach, whether or not constraints are used at the concrete level, can be contrasted with the two-level approach of commercial spatiotemporal query systems, such as Geographic Information Systems (GIS) [20] or spatial databases [19]. The main disadvantages of current GIS systems lie in their restrictions on data types and dimensions. Many approaches

rely on ad hoc data types and operations that may seem sufficient for current needs but fail when it comes to extending the structures with another dimension or achieving uniformity with another model. Specifically, spatial data operations are implemented only for the case of 2-dimensional data, and 3-dimensional data cannot be accommodated. Furthermore, the physical separation of alphanumeric and geometric data is carried up to the logical level, preventing uniform query declaration and processing for queries involving both alphanumeric and geometric data.

It has always been acknowledged that at the physical layer of constraint databases, other, more geometric, representations might be more useful. Already in [13], it was suggested that computational geometry algorithms could be used as primitives in the bottom-up query evaluations. Dedale (section 4.2), the leading constraint-based spatiotemporal database, also uses geometrical data representation; in their case, it is used at the *physical* level, translating between the geometric (vector) representation and the constraint representation when making use of the *constraint engine* [18].

It is time for geometric representations to come up from the physical level, and gain a foothold at the concrete level of constraint databases. The result is what we call *constraint-backed* database systems. In these systems, the *concrete* layer of constraint databases admits both constraints and geometric representations.

Positioning the geometric representation at the concrete, rather than the physical, level, has important consequences. In particular, the choice of representation can be delayed; both the database administrator and the query evaluation engine can make decisions regarding the best *concrete* representations for various relational schema with non-traditional attributes, from among those supported by the system.

Furthermore, when non-traditional data is represented geometrically rather than with constraints, geometric algorithms are used as primitives in the bottom-up query evaluations. These algorithms are invoked by the query processing engine. Neither the geometric data representation, nor the geometric algorithms, are visible to the user.

Note that constraints continue playing a major role in *constraint-backed* systems:

- They are needed for defining query semantics when e.g. proving query correctness.
- They provide default representation at the concrete level for those relations, whether imported or computed, where no other representation is readily available.
- They are used for data integration, as an "intermediate" representation between non-compatible systems.

The role of constraints for database interoperability has been discussed in [5]. While offering both of the above advantages associated with constraint databases, these systems are more *practical* and *efficient*:

- They are practical because they provide an adoption path for the commercial users (section 4.3), by allowing geometric representations to coexist with constraints.
- They are efficient because they provide a way to incorporate specialized geometric representations and optimized geometric algorithms for common spatiotemporal types of data, when these are more efficient than the general constraint approach 5.

4 Spatiotemporal Systems: Case Studies

In this section, we discuss three spatiotemporal query systems. Two are constraint-based: CQA/CDB and Dedale. Other constraint-based systems have been built, including MLPQ/GIS [16,17] and C^3 [3]; Space considerations prevent us from discussing them all. The third, the Oracle Spatial Data Option, is a leading object-relational system for managing spatial data.

4.1 CQA/CDB

CQA/CDB [8] is a "pure" constraint database where linear constraints are used to represent and process all data. Each constraint tuple is represented by a matrix of coefficients for the tuple's constraints. All data processing is performed over these matrices.

As in CQL [13,14], CQA/CDB has flat relational schema at the abstract logical level. Consider the following sample schema, based on the one in [18]:

```
LandUse(lname,x,y)
Flight(fname,t,x,y,a)
Country(cname,x,y,h)
```

where *lname* owns the parcel of land with coordinates x, y; *f*name the ID of a flight whose trajectory took it to altitude a over coordinates x, y at time t; *cname* is the name of a country with ground elevation h at coordinates x, y.

In this example, $\{x, y\}$ in *LandUse* are *related,* or not *independent,* in the sense of [4]. Similarly, $\{t, x, y, a\}$ are *related* in *Flight* and $\{x, y, h\}$ in *Country* are *related.* When attributes are related, this is indicated as a special *constraint* in the relational schema.

Furthermore, t serves as a key for $\{t, x, y, a\}$ in *Flight* [11]; that is, the value of $\{x, y, a\}$ for any relational tuple in the infinite extent of this relation is uniquely determined by the value of t. Similarly, (x, y) serves as a key for $\{x, y, h\}$ in *Country.* When keys are present, this would also be indicated in the schema, as a functional dependency.

Queries in CQA/CDB are based on relational operators, expressed as a list of operations rather than a single expression. Below are three queries and their CQA implementation.

1. *Over which location were the airplanes flying at time t_1?*

```
R0 = SELECT t=t1 from Flight
R1 = PROJECT R0 on fname,x,y
```

2. *Return the part of the parcels contained in rectangle Rect over x,y*

```
R0 = JOIN LandUse and Rect  (natural join on x,y)
```

3. *Return all Land parcels that have a point in Rect over x,y*

```
R0 = JOIN LandUse and Rect  (natural join on x,y)
R1 = PROJECT R0 on lname
R2 = JOIN R1 and LandUse
```

Note that the first step in query 3 is identical to query 2. This is due to the versatility of the *natural join:* it serves both as *intersection* and as *selection* [8].

CQA's list-of-operations notation is consistent with some database textbooks, such as [7]; also, it is analogous to 3-address codes for intermediate compiler languages [1], which have proven very convenient for compiler optimization.

While this is the only "pure" constraint database among our case studies, it is also the one that is furthest from commercial usability – probably not coincidentally.

4.2 Dedale

Dedale is a *constraint-based* spatial database system that, of all systems incorporating constraint database technology, comes the closest to potential commercialization. However, along the route towards eventual commercialization, several departures from pure constraint database technology were found necessary.

Dedale's nested data model represents a departure from CQL, the original constraint language of [13,14], which was flat. When the schema of a constraint relation has a group of *related* attributes, Dedale nests them together, introducing a single attribute to represent the group. For example, each of the relations in the sample schema from section 4.1 involves nesting:

```
LandUse(lname,geom[x,y])
Flight(fname,traj[t,x,y,a])
Country(cname,geom[x,y,h])
```

Note this nesting is done at the *abstract* logical level; user queries use explicit operators such as *unionnest* to nest and unnest the data. Here are the three queries from section 4.1, expressed in Dedale's algebraic notation:

1. *Over which location were the airplanes flying at time t_1?*

$$MAP_{\lambda X[X.fname, \pi_{x,y}(\sigma_{t=t_1}(X.traj))]}(Flight)$$

2. *Return the part of the parcels contained in rectangle Rect(x,y)*

$$MAP_{\lambda X[X.lname, X.geom \cap Rect]}(LandUse)$$

3.*Return all Land parcels that have a point in Rect over x,y* (where x ranges between x_{min} and x_{max} and y ranges between y_{min} and y_{max})

$$\pi_{lname,geom}(MAP_{\lambda X[X.lname,X.geom,\sigma_{x_{min}<x<x_{max} \wedge y_{min}<y<y_{max}}(X.geom)]}(LandUse))$$

Note that queries 2 and 3 no longer look the same, unlike in CQA/CDB.

Furthermore, Dedale limits query output to 2 spatiotemporal dimensions. An exception is when some spatiotemporal attributes serve as a *key* for others (such as t in *Flights,* or (x, y) in *Country*). In this case, the dependent (interpolated) attributes do *not* increase the total count of the dimensions.

Pure constraint databases such as CQA/CDB have no such limitation, and it would be unusual to introduce it for reasons of efficiency only. We believe that it exists because of Dedale's reliance on non-constraint representations for its implementation. For example, the *geom* nested attribute of the relation *Country* is represented internally as a *Triangulated Irregular Network* (TIN), and the *traj* nested attribute is represented as a set of finitely many sample points along the flight path [18].

4.3 Oracle Spatial Data Option

Until recently, Geographic Information Systems (GIS) were the leading commercial spatiotemporal systems. These GIS systems are *not* database systems per se:

1. They cannot manage non-geographic data.
2. They lack ad hoc querying capabilities; the user can either perform built-in operations or execute predefined queries.
3. They lack index structures for fast data access, performing all operations in memory.

Oracle's *Spatial Data Option* (SDO) extends Oracle for spatial data management, addressing all of these shortcomings:

1. It has the full power of a relational database, as well as support for spatial data, which is encapsulated as new datatypes within Oracle's object-relational framework. These are the same datatypes offered by ARC/Info, a leading GIS system.
2. Methods (operations) are supplied for the new spatial datatypes. These are also based on GIS operations.
3. Spatial data access structures based on *bounding boxes* are provided for off-line storage of spatial data.

Oracle SDO is typical of approaches being taken by object-relational databases to manage spatial data; another example of the newer GIS systems is ArcGIS. While avoiding the shortcomings of older GIS systems, this new approach is not perfect. The most significant problem is:

- The encapsulated spatial data is logically separate from the relational data stored in the same database.

All spatial objects, together with their *keys*. are stored in *spatial relations.* Any administrative information pertaining to these objects is stored in *regular relations* that refer to these spatial keys. This separation applies to query output as well, limiting the expressiveness of the system.

Constraint-backed systems are expected to be the next generation of spatiotemporal technology. They will providing the same advance over the current object-relational approaches as these represent over traditional GIS systems.

5 Disadvantages of Constraints

As the Dedale experience foreshadows (section 4.2), "pure" constraint databases will not prove as successful in practice as "mixed" approaches, where the abstract layer has relational semantics, but the implementation does not depend solely on constraints. In particular, the *concrete* layer of constraint databases should admit both constraints and geometric representations; we refer to such mixed approach as *constraint-backed.* In this section, we present some *practical* considerations why constraints are not always the best way to represent spatiotemporal data.

Unlike pure constraint databases, *constraint-backed database systems* support multiple data representations at the concrete database level, such as TINs or polylines in addition to constraints. However, they offer all the advantages of constraint databases, such as relational query semantics and heterogeneous querying of non-traditional relations with arbitrary schema. As a result, users are spared from learning scores of ad-hoc operators with seemingly arbitrary restrictions in their applicability.

The users' preference for conceptual simplicity also extends to visual data representation. Unfortunately constraints are not the answer here. Outside of research, we have yet to find users of spatiotemporal systems who prefer to view lists of constraints in output to their queries. Therefore, in commercial databases based on "pure" constraint technology, data must be continually translated between the *geometric* representation that is useful for data visualization and the *constraint* representation at the core of pure constraint databases.

The constraint-based data representation is far removed from the representation used for data input and (visual) output in commercial-level systems such as GIS, requiring costly conversions in each direction:

- The spatial data in Geographic Information Systems is normally obtained by *digitization* [20]. When digitizing a linear feature or a boundary of a region, streams of points and segments are generated. To use the constraint model, we must first convert these points to constraint representation.
- When displaying a feature as part of data visualization or query output, the reverse conversion must take place. In order to display a feature, its boundary points have to be computed from the constraints. The spatial outlines corresponding to each tuple must be found and combined together to obtain the feature boundary.

Furthermore, the constraint data model involves a lot of redundancy.

- Consider linear spatial features such as *roads, rivers,* or *hurricane trajectories.* These features can be viewed as consisting of many connected line segments, and their constraint representation consists of many constraint tuples – one for every segment. Each of these tuples involves three constraints: one for the line collinear with the segment, one for its starting point, and one for the ending point.
- Consider non-linear spatial features, such as *lakes, towns,* or *temperature zones.* These features can be viewed as bounded by a polyline consisting of many line segments, and concave in many places. The constraint data model requires us to represent this feature as a union of convex polyhedra, each of which is represented as a constraint tuple.

The representation of both types of features above involves two types of redundancies:

- Should the relation include attributes other than the spatial extent, these attributes are duplicated for each of the constraint tuples representing the same feature.
- The constraints representing the boundaries of each line segment or convex polyhedron are the same as for the tuples representing neighboring segments or polyhedra.

The first type of redundancy cannot be avoided when a flat relational model is used at the concrete level. Dedale's decision to depart from the relational model and use the *nested* model instead (section 4.2) avoids this redundancy. While we agree with the need for nesting, it is not clear whether it should take place at the abstract level, as in Dedale, or at the concrete, so the abstract representation can remain flat as in CQA/CDB (section 4.1).

The second type of redundancy is also unavoidable as long as a pure constraint-based representation is used. Dedale's solution here is to omit the constraints that specify the boundaries of each line segment. Instead, the bounding box of the segment serves to define its boundaries.

To conclude, we note that query evaluation can be performed directly over the geometrical representation, by applying methods from computational geometry. This avoids having to translate the data into constraint format, and sometimes it even simpler than the corresponding constraint operation:

Example 1. If two-dimensional regions are stored directly as a sequence of points that outline it, a region's projection onto either of the dimensions can be obtained by taking the corresponding coordinate of each point, and finding the extrema of these values.

6 Summary

We summarize our discussion with the following list of points:

- Our aim is for *constraint-backed database systems* that offer all the advantages of constraint databases such as relational query semantics and heterogeneous querying of spatiotemporal and other non-traditional relations with arbitrary schema. Unlike pure constraint databases, these systems support multiple data representation at the concrete database level in addition to constraints.
- Like pure constraint databases, constraint-backed database systems are *three-tiered,* with *abstract, concrete,* and *physical* layers. In contrast, relational databases as well as spatial are two-tiered, with the *abstract* and *concrete* layers merged into a single *logical* layer.
- In constraint-backed systems, constraints are needed for defining query semantics when e.g. proving query correctness. Also, they provide default representation at the concrete level for those relations, where no other representation is readily available. Finally, they are useful for data integration, as an "intermediate" representation between non-compatible systems.
- It is the job of the database administrator, or the query processing engine if the relation is computed, to choose the best *concrete* representation for any given relation with non-traditional attributes, among those that are supported by the database system. This choice is transparent to the users.
- When non-traditional data is represented geometrically rather than with constraints, geometric algorithms must be used as primitives in the bottom-up query evaluations. These algorithms are invoked by the query processing engine, and are invisible to the user.

This list of points is not to be viewed as a spec for system-building. Query processing engines for three-tier constraint-backed systems are necessarily more sophisticated than for pure constraint databases, or for traditional spatial databases. Many research issues must be addressed before constraint-backed database systems are ready for commercialization, or even for serious prototyping. For example, there are consequences for query optimization and database interoperability. We are confident that these research issues will in time be addressed, and that constraint-backed database systems will someday be the technology of choice for all spatiotemporal data processing.

References

1. A. Aho, R. Sethi, J. Ullman *Compilers: Principles, Techniques, and Tools.* Addison Wesley, 1988.
2. M. Atkinson, F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier, S. Zdonik. The Object-Oriented Database System Manifesto. In *Proc. ACM SIGMOD,* 1990.
3. A. Brodsky, V. E. Segal, and P. A. Exarkopoulo. The CCUBE constraint object-oriented database system. *CONSTRAINTS, An International Journal,* 51(1), pp. 26–52, 1997.

4. Jan Chomicki, Dina Goldin, Gabriel Kuper, and David Toman. Variable Independence in Constraint Databases. *IEEE Transactions on Knowledge and Data Engineering,* 2003.
5. J. Chomicki, P. Revesz. Constraint-Based Interoperability of Spatiotemporal Databases. *Geoinformatica,* 3:3, pp. 211-243, 1999.
6. E.F. Codd. A Relational Model for Large Shared Data Banks. In *Communications of the ACM* 13(6), pp. 377–387, 1970.
7. R. Elmasri and S.B. Navathe. *Fundamentals of Database Systems,* 3rd edition. Addison Wesley, 2000.
8. D. Goldin, A. Kutlu, M. Song, F. Yang. The Constraint Database Framework: lessons learned from CQA/CDB. In *Proc. ICDE 2003 (Int'l Conf. on Database Engineering),* Bangalore India, March 2003.
9. D.Q. Goldin and P.C. Kanellakis. Constraint Query Algebras. *Constraints Journal,* 1(1/2), pp. 45–83, 1996.
10. S. Grumbach, P. Rigaux, L. Segoufin. The DEDALE System for Complex Spatial Queries. In *Proc. ACM SIGMOD,* June 1998.
11. S. Grumbach, P. Rigaux, L. Segoufin. Manipulating Interpolated Data is Easier than You Thought. In *Proc. Very Large Database Conference* (VLDB), 2000.
12. R. Guting, M. Bohlen, M, Erwig, C. Jensen, N. Lorentzos, M. Schneider, M. Vazirgiannis. A Foundation for Representing and Querying Moving Objects. *ACM Transactions on Database Systems* (TODS) 25(1), pp. 1-42, 2000.
13. P.C. Kanellakis, G.M. Kuper, and P.Z. Revesz. Constraint Query Languages. *Proc. 9th ACM PODS Symposium on the Principles of Database Systems* (PODS), Nashville Tennessee USA, pp. 299–314, March 1990.
14. P.C. Kanellakis, G.M. Kuper, and P.Z. Revesz. Constraint Query Languages. *J. Computer and System Sciences,* 51(1), pp. 26–52, 1995.
15. *Open GIS Consortium.* URL: http://www.opengis.org/
16. Revesz, P., Chen, R., Kanjamala, P., Li, Y., Liu, Y., Wang, Y. The MLPQ/GIS Constraint Database System. *ACM SIGMOD Record* 29:2, 2000.
17. P. Revesz. *Introduction to Constraint Databases.* Springer Verlag, New York, 2002.
18. P. Rigaux, M. Scholl, L. Segoufin, S. Grumbach. Building a Constraint-Based Spatial Database System: Model, Languages, and Implementation. To appear in *Information System Journal,* 2003.
19. P. Rigaux, M. Scholl, and A. Voisard. *Spatial Databases with Application to GIS.* Academic Press, 2002.
20. M. F. Worboys. *GIS: A Computing Perspective.* Taylor & Francis, 1995.

Integrating Constraint and Relational Database Systems

Mengchu Cai

IBM Silicon Valley Laboratory
San Jose, CA, USA
mcai@us.ibm.com

Abstract. Constraint databases are a generalization of relational databases and can finitely represent infinite relations. The high expressiveness of constraint databases makes them a promising data model for representing spatial and spatiotemporal information. On the other hand, relational database systems are predominantly used for traditional applications and have a proven record of providing efficient querying and management of business data. It is more cost-effective to integrate constraint databases with the existing relational database systems than to build a completely new constraint database system from scratch. This paper proposes a way to integrate the constraint data model and query languages with a relational database system. It introduces a new relational data type to represent constraint data. It also provides several new relational database operators to query the constraint data.

1 Introduction

Constraint databases [11,15] are a generalization of relational databases and provide uniform representation for diverse data. The capability to finitely represent infinite relations makes constraint databases a promising data model to represent spatio-temporal information and has drawn a growing interest in the database systems community.

On the other hand, the relational databases enjoy great success in the business world and have a proven record for providing efficient and cost-effective management of structured data and business transactions. The SQL query language is used predominantly in a huge number of applications. It would be desirable and cost effective to combine the expressiveness of constraint databases and the reliability and efficiency of existing relational databases.

The basic concepts of relational databases are tables and columns. Each table contains a finite set of columns and each column has a declared data type to specify what kind of values can be stored in that column. A natural idea for storing constraint data in relational databases is to store them in columns of tables and introduce a new type to describe the columns. A relatively easy way for this is to use user-defined types and user-defined functions [19]. However, in this way, the relational database engines do not understand the semantics of the constraint data, and all operations on constraint data are handled by calling

B. Kuijpers and P. Revesz (Eds.): CDB 2004, LNCS 3074, pp. 173–180, 2004.

the user-defined functions which are applications to the database systems. The communication overhead between the database engine and those user-defined functions could be a big hurdle to achieve high query performance. And there are no user-defined aggregate functions in SQL [19] yet. In this paper, we introduce constraint formula as a new built-in SQL data type called CONSTRAINT in the relational databases. We also introduce a set of built-in operators including aggregate operators on constraint formulae.

This paper is organized as follows. In Section 2, we describe the *constraint data type* and describe the representation of the constraint data in relational databases at the logical level. In Section 3, we describe the internal storage format for the constraint data. In Section 4, we introduce several functions to invoke the constraint query language from an SQL context and get the result back to the relational system. Finally, Section 5 covers related and future work.

2 Constraint Data Type

Constraint databases represent data by a set of disjunctive normal form logical formulae. For example, the spatial extent of the city shown in Figure 1 can be represented in constraint databases by the following formula:

$$x \geq 0, \;\; y \geq 0, \;\; y \leq 5, \;\; x + y \leq 10$$

where the commas denote conjunctions of atomic formulas. In this case the atomic formulas are linear inequality constraints, which are commonly used in prototype constraint database systems.

The natural semantics of the above formula is an infinite set of (x, y) points in the real plane that satisfy the formula.

Spatio-temporal objects can be represented similarly by using an extra variable t for time. For example, suppose the front line of a moving rain area is moving southeast at the speed 2 miles per minute. It can be represented as follows:

$$x - y \leq 7 - 2t, \;\; t \geq 0, \;\; t \leq 5$$

To integrate such constraint data into relational database systems, we propose the introduction of constraint formulae as a basic SQL data type called *constraint* in relational databases. We can use this type to define a column and store constraint data as an instance of *constraint* data type in a relational table.

Example 1. In the following, we define a relational table *Cities* to store the following information about cities: the name of the city as a character string, the country the city belongs to as another character string, and the spatial extent of the city as a *constraint* data type with (x, y) dimensions.

```
CREATE TABLE cities( name    VARCHAR(20) NOT NULL,
                     country VARCHAR(15) NOT NULL,
                     extent  CONSTRAINT(x,y) );
```

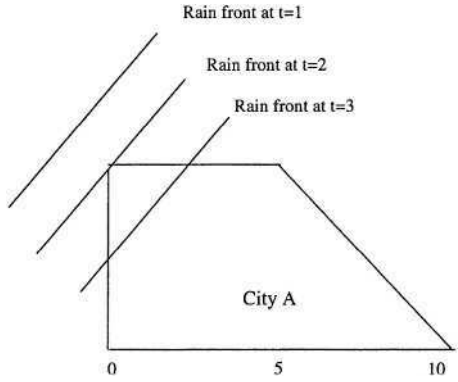

Fig. 1. An instance of a city and rain

The following are example data that represents City A in Figure 1. We use string representation of the CONSTRAINT data type instance.

Cities

Name	Country	Extent
A	USA	$x \geq 0,\ y \geq 0,\ y \leq 5,\ x + y \leq 10$
Paris	France	...
⋮	⋮	⋮

Example 2. Now let us represent the rainy area spatio-temporal information using a *constraint* data type. We define a relational table *Rains* with an *id* as an integer and an *area* of type *constraint(x,y,t)*.

```
CREATE TABLE Rains ( id  INTEGER NOT NULL,
                     area CONSTRAINT(x,y,t) );
```

The following is an instance of the *Rains* relation corresponding to Figure 1.

Rains

Id	Area
1	$x - y \leq 7 - 2t,\ t \geq 0,\ t \leq 5$
⋮	⋮

3 Relational Storage of Constraints

The simplest way to store instances of *constraint data type* is to store their string representations as instances of *varchar* or *clob* data type. However, we

need to parse the string every time we access the instance, and that would not be efficient.

An alternative way is to store the parsed format of the instance in the base table. Each instance of n-dimensional constraint data type can be rewritten into a disjunction of m terms, where each term is a conjunction of k linear inequality atomic constraints with n variables. Figure 2 shows an example of this format. Usually the size of a row is restricted by the page size used in the database system. Different database systems support different page sizes, which could be 4K, 8K, 16K or 32K bytes. Storing instances of constraint data type within the base table could limit the size of the instances.

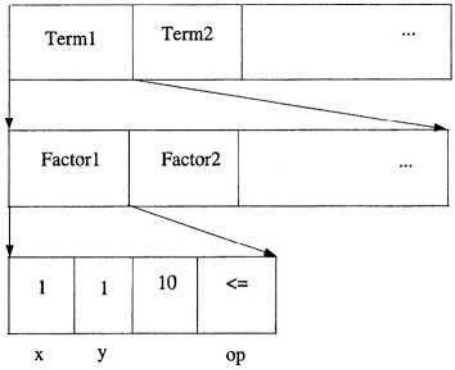

Fig. 2. Store format

Another approach is to store the instance in an auxiliary table which is anchored at the base table. In column of CONSTRAINT data type of base table, we store some meta data like the number of dimensions and so on. The real content is shredded into another table. The system will add an extra column called row id in both the base table and the auxiliary table to associate them together. An index on the auxiliary table on the row id column will be created automatically to speed up the access of the CONSTRAINT instance. Let us consider Example 1 again. The table *Cities* will be split into two tables as follows. The extent column of the table Cities stores the number 2, which is the number of dimensions. Its real content is stored in an auxiliary table of seven columns and four rows. Each row in the auxiliary table represents a factor in the formula. The data in the columns x and y are the coefficients of the corresponding variables in the formula, the *op* column stores the kind of operator $(=, >, <, \geq, \leq)$ used in atomic constraint, and the *fid* column is the id of the atomic constraint. The combination of base row id, term and fid can uniquely identify a row in the auxiliary table.

Name	State	Extent	ROW ID
A	CA	2	A0001

Base ROW ID	term	fid	op	x	y	constant
A0001	1	1	$\geq$	1	0	0
A0001	1	2	$\geq$	0	1	0
A0001	1	3	$\leq$	0	1	5
A0001	1	4	$\leq$	1	1	10

The store formats are implementation details. In Section 4, we just use a string representation for the sake of a high level description of the query language.

4 Querying Constraint Data Type

Querying a relational database usually requires two main steps: first, to find the relevant rows, and second to construct a new instance from those rows. The SQL query language can be used to query the tables by the conditions on the non-constraint columns. However, ordinary SQL cannot query information inside a constraint data type instance. That is what constraint query languages are designed for. What we have to achieve now is a way to integrate the two, so that we can invoke constraint query language from SQL, and also to provide information from the relational environment to the constraint evaluation engine. Due to the complexity to extend the SQL parser to understand both SQL and constraint query language, we propose a set of pseudo functions to wrap the constraint query expressions and make them like regular functions in an SQL statement. We can separate the SQL parser and the parser and processor for the constraint query language and have a more modularized architecture.

The simplest operation is to query an instance of the constraint data type applied on instances of constraint data type and return another instance of the constraint data type. We propose a scalar function called *CQuery* for this operation. The signature of this function is described bellow:

1) CQuery(constraint query, arguments)$\rightarrow$ constraint

This function takes as input a constraint query string and a variable number of arguments which could be a constraint type. It invokes the constraint evaluation engine to evaluate the constraint query with the arguments passed in and returns an instance of the constraint data type. This function can be used within an SQL statement where a scalar function can appear. Example 3 shows the use of CQuery in the SELECT clause.

Example 3. Let us consider again the Cities and the Rains tables. Suppose the question is in which part of city A has rained at time $t = 3$. This can be found using the following extended SQL query:

```
SELECT CQuery('C.Extent(x,y), R.Area(x,y,3)') as rain_area
FROM   cities C, rains R
WHERE C.name = 'A' AND R.id = 1;
```

The result is the following:

rain_area
$x \geq 0,\ y \leq 5,\ x - y \leq 1$

Union and intersection are two very useful operations for many queries. In a constraint query language, it is straightforward to express the union or intersection of two instances of the constraint data type. Example 3 is an example of intersection.

Below we also propose two new aggregate functions, *CAggUnion* and *CAggIntersect* that are more general than simple union and intersection, because they allow grouping in the SQL style.

2) CAggUnion(constraint)→ constraint

This function computes the union of the set of instances of constraint type for each group.

Example 4. Find the raining city areas in each country at time $t = 3$.

```
SELECT    C.country, CAggUnion(CQuery('C.extent (x,y), R.area (x,y,3)'))
FROM      cities C, rains R
GROUP BY C.country;
```

3) CAggInterect(constraint)→constraint

This function computes the intersection of the set of instances of a constraint type for each group.

Example 5. Find the areas where it rains all the time between $t = 1$ and $t = 5$.

```
SELECT CAggIntersect(CQuery('R.area(x, y, t), t ≥ 1, t ≤ 5'))
FROM   rains R;
```

Some applications, such as decision support systems, may be more interested in whether it rains at a certain time or not than in the exact boundary of a rainy area. Therefore, we introduce a new predicate (CExists) to test whether the result of a constraint query can be satisfied.

4) CExists(constraint query, arguments)→Boolean

CExists is a predicate that evaluates a constraint query and returns *true* if the result of the query is a set of constraints that can be satisfied and returns *false* otherwise.

Example 6. Find the names of the cities where it rained sometime before $t = 4$.

This query can be expressed by the following SELECT statement. In this example, an *as* clause is added to rename the column name extent and area to *P* and *Q,* which are then used as the relation names inside the constraint query string.

```
SELECT C.name
FROM   cities C, rains R
WHERE CExists('P(x, y), Q(x, y, t), t ≤ 4',
        C.extent AS P, R.area AS Q))
```

5 Related Work and Conclusions

After the pioneering work of Kanellakis et al. [11] proposed the constraint data model, extensive research was done on the expressiveness [1], complexity, optimization [2] and indexing of constraint databases [7]. There are also several prototype implementations of constraint database systems. DISCO [5] and MLPQ [12,17,18] are prototypes built from scratch and store data directly in file systems.

Brodsky et al. [3,4] proposed to extend object-oriented databases by adding a new data type CST for constraint spatiotemporal objects. [3] focused on extending high level OQL to support constraint query language. [4] defined the constraint data type as a C++ class and explored optimization level query language. The DEDALE system Grumbach et al. [10] extends the set model by adding a set type based on linear constraints and defined many operators on the extended model. [3,4,10] implemented their prototype systems on top of an object-oriented DBMS.

Chomicki et al. [9] proposed a way to animate spatiotemporal objects in constraint database. [6] proposed a data model that combines querying and animating spatiotemporal objects. [14] describes spatio-temporal interpolation methods whose results can be represented in constraint databases. A good introduction to the area of constraint databases giving more discussion on related work is the textbook by Revesz [15]. Kuper et al. [13] is another book on constraint databases.

This paper proposed to integrate constraint data model and query language into relational databases. It describes how to store instances of constraint database in relational databases. Instead of introducing equivalent operators to SQL, it introduces several operators to invoke constraint query languages from SQL and minimize the impact on the existing SQL system. An interesting issue could be combining the optimization and index technologies for constraint databases and relational databases. It is an open problem whether some of the more recent spatiotemporal data models that extend the original constraint data model (e.g. [8,16]) can be also integrated into relational database systems in a similar way.

References

1. M. Benedikt, G. Dong, L. Libkin and L. Wong: Relational Expressive Power of Constraint Query Languages. *Proc. 15th ACM Symp. on Principles of Database Systems,* 5-16, 1996.
2. A. Brodsky, J. Jaffar, M.J. Maher: Toward Practical Constraint Databases. *Proc. 19^{th} VLDB,* 322-331, 1993
3. A. Brodsky, Y. Kornatzky: The Lyric Language: Querying Constraint Objects. *Proc. SIGMOD,* 35-46, 1995.
4. A. Brodsky and V. Segal: The C^3 Constraint Object-Oriented Database System: An Overview. *CDB 1997,* 134-159.
5. J. Byon and P. Revesz: DISCO: A Constraint Database with Sets. *Proc. Workshop on Constraint Databases and Applications,* Springer LNCS 1034, pp. 68-83, 1995.

6. M. Cai, D. Keshwani, P. Revesz: Parametric Rectangles: A Model for Querying and Animation of Spatiotemporal Databases, *Proc. Seventh Conference on Extending Database Technology (EDBT),* Springer LNCS 1777, pp. 430-444, 2000.
7. M. Cai, P. Revesz: Parametric R-tree: An Index Structure for Moving Objects, *Proc. Tenth International Conference on Management of Data,* McGraw Hill, pp. 57–64, Pune, India, 2000.
8. J. Chomicki, S. Haesevoets, B. Kuijpers, and P. Revesz: Classes of Spatiotemporal Objects and their Closure Properties, *Annals of Mathematics and Artificial Intelligence,* vol. 39, no. 4, pp. 431-461, 2003.
9. J. Chomicki, Y. Liu, P. Revesz: Animation of Spatiotemporal Databases, *Workshop on Spatio-Temporal Database Management,* Springer LNCS, Edinburgh, Scotland, 1999.
10. S. Grumbach, P. Rigaux, and L. Segoufin: The DEDALE system for Complex Spatial Queries, *SIGMOD,* 1998.
11. P. Kanellakis, G. Kuper, and P. Revesz: Constraint query languages. *Journal of Computer of System Sciences,* 51:26-52, 1995.
12. P. Kanjamala, P. Revesz and Y. Wang: MLPQ/GIS: A Geographic Information System using Linear Constraint Databases, *Proc. Ninth International Conference on Management of Data,* pp. 389-392, 1998.
13. G. Kuper, L. Libkin and J. Paredaens: *Constraint Databases,* Springer, 2000.
14. L. Li and P. Revesz: Interpolation Methods for Spatiotemporal Geographic Data, *Journal of Computers, Environment, and Urban Systems,* vol. 28, no. 3, pp. 201-227, 2004.
15. P. Revesz: *Introduction to Constraint Databases.* Springer, New York, 2002.
16. P. Revesz and M. Cai: Efficient Querying of Periodic Spatiotemporal Objects, *Annals of Mathematics and Artificial Intelligence,* vol. 36, no. 4, pp. 437-457, 2002.
17. P. Revesz, R. Chen, P. Kanjamala, Y. Li, Y. Liu, and Y. Wang: The MLPQ/GIS Constraint Database System, *SIGMOD,* 2000.
18. P. Revesz and Y. Li: MLPQ: A Linear Constraint Database System with Operators, *Proc. 1st International Database Engineering and Application Symposium,* pp. 132-137, 1997.
19. ISO/IEC 9075-2:2003 (SQL/Foundation).

Author Index

Cai, Mengchu 173
Ceballos Guerrero, Rafael 74
Chomicki, Jan 128
Choueiry, Berthe Y. 143

Geerts, Floris 40
Goldin, Dina Q. 161
Gómez López, Maria Teresa 74

Haesevoets, Sofie 52
Heintz, Joos 1

Kuijpers, Bart 1

Lal, Anagh 143
Leung, Carson Kai-Sang 112
Li, Lixin 25
Li, Youming 25

Martínez Gasca, Rafael 74

Piltner, Reinhard 25

Ramanathan, Viswanathan 88
Revesz, Peter 88

Valle Sevilla, Carmelo del 74