Jean-François Boulicaut and Sašo Džeroski, editors

Proceedings of the 2nd International Workshop on Knowledge Discovery in Inductive Databases (KDID-2003)

Cavtat/Dubrovnik, Croatia, 22 September 2003

The 14th European Conference on Machine Learning and
The 7th European Conference on Principles and Practice
of Knowledge Discovery in Databases
(ECML/PKDD-2003)

Foreword

The 2nd International Workshop on Knowledge Discovery in Inductive Databases (KDID-2003) was held in Cavtat/Dubrovnik, Croatia, on 22 September 2003 as part of the 14th European Conference on Machine Learning and the 7th European Conference on Principles and Practice of Knowledge Discovery in Databases (ECML/PKDD-2003).

Ever since the start of the field of data mining, it has been realized that the data mining process should be supported by database technology. In recent years, this idea has been formalized in the concept of inductive databases introduced by Imielinski and Mannila in a seminal paper that appeared in CACM 1996 (Vol. 39, Issue 11, pages 58-64). Inductive databases are databases that, in addition to data, also contain generalizations, i.e., patterns, or models extracted from the data. Within the inductive database framework KDD is modelled as an interactive process in which users can query both data and patterns/models to gain insight about the data. To this aim a so-called inductive query language is used. Very often inductive queries impose constraints on the patterns/models of interest (e.g. w.r.t. frequency, syntax, generality, accuracy, significance). This constraint-based approach to data mining is also closely related to the issue of declarative bias in machine learning, e.g., to syntactic bias, which imposes certain (syntactic) constraints on the patterns learned.

Today, a number of specialized inductive query languages have already been proposed and implemented (e.g., MINE RULE by Meo et al., MSQL by Imielinski et al., DMQL by Han et al.). Most of these inductive query languages extend SQL and concern mainly local patterns like descriptive rules. Recently the database community has also become interested in the use of XML (eXtensible Markup Language), which has rapidly become an important standard for representing and exchanging information through its applications. Thus, developing XML aware data mining techniques and query languages is also a significant part of the research. Beside these query language proposals, a large body of research work looks for the key concepts to be used in inductive databases. On the one hand, this includes the many types of constraints that have been employed, and on the other hand, the many efficient algorithms for solving constraint-based queries (e.g., the level-wise algorithm, the version space algorithm, condensed representations). Also, there exists a large body of research on database techniques that enable and optimize the data mining process. This includes various new database primitives that have been suggested, e.g., to efficiently compute statistics needed by data mining algorithms.

Despite these many contributions, we are still far away from a deep understanding of the issues in inductive databases. In this respect, one can see an analogy with the state-of-the-art in databases in the early seventies. Today, there exist many different approaches and implementations of data mining systems but our theoretical understanding is still imperfect. In a sense, we are still looking for the equivalent of Codds relational database theory for data mining.

Given the many recent developments in the area of inductive databases and constraint-based mining, it was timely to organise this second international workshop. The first workshop of the series, KDID-02, organised in conjunction with ECML/PKDD-2002 in Helsinki, Finland, (Helsini), attracted around 35 participants and many interesting contributions; see the workshop web site at `http://www.cinq-project.org/ecmlpkdd2002/` for details. The aim of this workshop was to bring together researchers and practitioners of data mining interested in inductive databases constraint-based data mining and data mining query languages.

A non-exclusive list of topics from the call for papers is as follows:

- Query languages for data mining
- Constraint-based data mining
- Pattern domains and primitives for data mining
- Declarative bias formalisms in machine learning
- Version spaces and boundary set representations
- Database support, coupling and primitives for data mining
- Coupling of database and data mining systems
- Integration of database languages such as SQL and XML with data mining
- Specific and application-oriented inductive databases

The scientific program of the workshop included 10 papers and an invited talk by Minos Garofalakis. We wish to thank the invited speaker, all the authors who submitted their papers to KDID-2003, the PC members for their help in the reviewing process, and the ECML/PKDD organizing committee for help with local organization.

Lyon/Ljubljana — Jean-François Boulicaut
Sašo Džeroski

July 2003

This workshop was supported by the European Union project cInQ IST-2000-26469 funded by the Future and Emerging Technologies arm of the IST Programme. The partners of cInQ (consortium on discovering knowledge with Inductive Queries) are Jozef Stefan Institute (Slovenia), Nokia Research Center (Helsinki, Finland), University of Freiburg (Germany), Politecnico di Milano (Italy), University of Torino (Italy), and INSA Lyon (France).

Chairs

Jean-François Boulicaut
INSA Lyon, LIRIS CNRS FRE 2672, Batiment Blaise Pascal
F-69621 Villeurbanne cedex, France
Email: Jean-Francois.Boulicaut@insa-lyon.fr,
URL: http://lisi.insa-lyon.fr/~jfboulic/

Sašo Džeroski
Department of Intelligent Systems, Jožef Stefan Institute
Jamova 39, SI-1000 Ljubljana, Slovenia
Email: Saso.Dzeroski@ijs.si, URL: http://www-ai.ijs.si/SasoDzeroski/

Organizing Committee

Jean-François Boulicaut, co-chair (INSA-Lyon, France)
Luc De Raedt (University of Freiburg, Germany)
Sašo Džeroski (Jožef Stefan Institute, Slovenia)
Mika Klemettinen (Nokia Research Center, Helsinki, Finland)
Rosa Meo (Universitá di Torino, Italy)

Program Committee

Elena Baralis (Politecnico di Torino, Italy)
Marco Botta (Universitá di Torino, Italy)
Jean-François Boulicaut, co-chair (INSA-Lyon, France)
Stefano Ceri (Politecnico di Milano, Italy)
Luc De Raedt (University of Freiburg, Germany)
Sašo Džeroski, co-chair (Jožef Stefan Institute, Slovenia)
Bart Goethals (HIIT-BRU, Finland)
Mika Klemettinen (Nokia Research Center, Finland)
Krzysztof Koperski (Insightful Corporation, USA)
Stefan Kramer (Technische Universität Muenchen, Germany)
Pier Luca Lanzi (Politecnico di Milano, Italy)
Dominique Laurent (University of Tours, France)
Giuseppe Manco (ICAR-CNR, Italy)
Heikki Mannila (Helsinki University of Technology / HIIT-BRU, Finland)
Rosa Meo (Universitá di Torino, Italy)
Jian Pei (SUNY at Buffalo, USA)
Christophe Rigotti (INSA-Lyon, France)
Lorenza Saitta (Amedeo Avogadro University, Eastern Piedmont, Italy)
Hannu TT Toivonen (University of Helsinki, Finland)
Jeff D. Ullman (Stanford University, USA)
Jiong Yang (University of Illinois at Urbana Champaign, USA)
Philip S. Yu (IBM T.J. Watson Research Center, USA)
Mohammed Zaki (Rensselaer Polytechnic Institute, USA)

Table of Contents

Constraint-Based Model Mining: Algorithms and Applications (invited talk)

Minos Garofalakis

Bell Labs
600 Mountain Avenue
Murray Hill, NJ 07974, USA.
minos@research.bell-labs.com http://www.bell-labs.com/ minos/

Abstract. Extracting interesting models from large amounts of data is typically a rather expensive, iterative process that is often characterized by a lack of focus on the specific models of interest to individual users. Exploiting user-defined model constraints during the mining process can significantly reduce the required computational effort and ensure system performance that is commensurate with the level of user focus. Attaining such performance goals, however, is not straightforward, often mandating the design of novel mining algorithms that can make effective use of the model constraints. In this talk, I will try to provide an overview of recent work and results on scalable, constraint-based data-mining algorithms. I will also discuss some interesting applications of such techniques, including our own work on model-based semantic compression tools for massive data tables. Finally, I will provide some (personally-biased) directions for future research in this area and (time permitting) touch upon problems in the evolving field of data-stream mining.

A framework for frequent sequence mining under generalized regular expression constraints

Hunor Albert-Lorincz and Jean-François Boulicaut

INSA Lyon LIRIS CNRS FRE 2672 - Bâtiment Blaise Pascal
F-69621 Villeurbanne Cedex, France
{Hunor.Albert-Lorincz@insalien.org, Jean-Francois.Boulicaut@ insa-lyon.fr}

Abstract. This paper provides a framework for the extraction of frequent sequences satisfying a given regular expression (RE) constraint. We take advantage of the information contained in the hierarchical representation of an RE by abstract syntax trees (AST). Interestingly, pruning can be based on the anti-monotonicity of the minimal frequency constraint, but also on the RE constraint, even though this latter is generally not anti-monotonic. The AST representation enables to examine the decomposition the RE and to choose dynamically an adequate extraction method according to the local selectivity of the sub REs. Our algorithm, RE-Hackle, explores only the candidate space spanned over the regular expression, and prunes it at each level. Due to the dynamic choice of the exploration method, this algorithm surpasses its predecessors. We provide an experimental validation on both synthetic data and a real genomic sequence database. Furthermore, we show how this framework can be extended to regular expressions with variables providing context-sensitive specification of the desired sequences.

1. Introduction

Frequent sequential pattern mining in a database of sequences is an important data mining technique [1,11]. Its application domains span from the analysis of biological data to the discovery of WWW navigation paths or alarm analysis. In most of these applications, the lack of user control in specifying the interesting patterns beforehand leads to intractable extraction or costly post-processing phases. Considering other user-defined constraints, in conjunction with the minimal frequency constraint, has been proved useful [10,3,12,9]. Not only it limits the number of returned patterns but also it can reduce extraction cost to an acceptable extent. In this paper, we consider constraints expressed by regular expressions (RE). Thanks to REs, the user can specify the language of desired patterns in a rather flexible way. We consider that a *sequence* (or string) is an ordered list of items taken from a finite alphabet $\mathcal{A}$. ||S|| denotes the length of S, i.e., the number of items it contains. A database of sequences (D) is an unordered collection of sequences. $S' = s'_1 s'_2 ... s'_m$ is called a sub-sequence of $S = s_1 s_2 ... s_n$ ($m<n$) if $\exists$ k s.t. $s'_1 s'_2 ... s'_m = s_{k+1} s_{k+2} ... s_{k+m}$. The frequency of a sequence S Freq(S,D) is the number of the sequences S' in D s.t. S is a sub-sequence of S'.

Problem statement: Given a database D, a regular expression E and a positive integer *minsup*, find all the sequences S which satisfy the following properties:

- Freq(S,D) ≥ *minsup*
- $S \in \mathcal{L}(E)$, $\mathcal{L}(E)$ being the language specified by the regular expression E.

Constraint-based mining is difficult. Efficient algorithms are often ad-hoc, i.e., they are optimized for specific constraints and/or a specific kind of data. Several algorithms exploit efficiently the minimal frequency constraint thanks to its anti-monotonicity [1,6,4,8,11]. Pushing constraints that involve other user-defined constraints has been studied as well, e.g., in [3,11,9,7]. A RE-constraint is generally neither anti-monotonic nor monotonic and thus can not be used directly for pruning the search space. Furthermore, if a non anti-monotonic (nAM) constraint is "pushed" inside an extraction algorithm, the requirement that the sequences must satisfy both the minimal frequency and the nAM constraint can lack of pruning [3]. To tackle RE-constraints, the authors of the SPIRIT algorithms have proposed in [3] several relaxations of RE constraints. These relaxations have led to four ad-hoc algorithms that use different pruning strategies and perform unequally depending on the selectivity[1] of the constraint. In other terms, the choice of a given SPIRIT algorithm must be based on the selectivity of the RE constraint which is a priori unknown. We would like a robust algorithm which depends weakly of the selectivity of the constraint and the characteristics of the data, i.e., an algorithm which would consider the individual selectivity of the sub-expressions and choose the best pruning strategy during the extraction according to the sequences in the database. This would be a major step towards efficient sequence mining for many application domains.

We proposed in [2] the RE-Hackle (Regular Expression-Highly Adaptive Local Extraction) framework. It is based on a hierarchical abstract syntax tree (AST) of the RE that can collect much more information on the local properties of the constraint than the previous methods based on Finite State Automata [3]. The collected information is used to examine each sub-expression of the initial RE and the RE-Hackle algorithm chooses the extraction strategy to favor pruning on the minimal frequency constraint or on the RE-constraint. For that purpose, candidate sequences are assembled during a bottom-up processing of the AST. Each node of this tree, the Hackle-tree, encodes a part of the RE and, when the algorithm reaches a node, it has information about the frequent sequences generated by its sub-tree. The number of these sequences, the structure of the tree in the neighborhood of the node and global information such as the cost per candidate evaluation and per database scan can be used to determine a candidate generation strategy and to optimize the execution time.

The contribution of this paper is threefold. We provide an introduction to the RE-Hackle algorithm defined in [2] and we introduce a new optimization technique based on Hackle-trees. Also, we provide experimental results on both synthetic data and real biological data that were missing from [2]. It shows that our algorithm adapts dynamically its pruning strategy and tends to take the shape of the best SPIRIT algorithm without any prior knowledge of constraint selectivity. Finally, we show that the framework can be extended to regular expressions with variables. Adding variables clearly increases the expressive power of the exploration language and reaches beyond the scope of context free grammars.

[1] Selectivity is an intuitive characterization of REs which is roughly speaking inversely proportional to the number of sequences in the database that match the initial constraint.

The paper is organized as follows. Section 2 provides the needed definitions. The RE-Hackle basic algorithm is described in Section 3. An original optimization technique is described in Section 4. Section 5 overviews the experimental results and it compares RE-Hackle to our fair implementation of two SPIRIT algorithms. Section 6 introduces an extension of the framework to regular expressions with variables. Section 7 concludes.

2. Definitions

A *RE constraint* in a kind of a regular expression built over an alphabet of sequences using the following operators: union (denoted +), concatenation (denoted $¤_k$ and sometimes "." when k is 0) and Kleene closure (denoted *). The empty set and the empty sequence are noted Ø and ε.

The *k-telescoped concatenation* of two sequences $S=s_1 s_2 ...s_{||s||}$ and $P= p_1 p_2 ...p_{||P||}$ is a new sequence. This operator requires that the sequences overlap in k positions. When k is zero, we get the usual concatenation. It is used for candidate generation.

$$Op¤_k(S, P)= S ¤_k P = \{ s_1 s_2 ... s_{||s||-k} p_1 p_2 ...p_{||p||}\} \text{ if for all } 0<j\leq k \text{ we have}$$

$p_j=s_{||s||-k+j}$ and $||S|| > k$ and $||P|| > k$. If $\exists\ 0<j\leq k$ s.t. $p_j<>s_{||s||-k+j}$ $Op¤_k(S, P)=ε$.

Concatenating two sets of sequences $\{S_1, S_2, \ldots, S_n\}$ and $\{P_1, P_2, \ldots, P_m\}$ gives a new set of sequences that contains all the sequences resulting from the Cartesian product of the sequences from the two sets.

$$\{S_1, \ldots, S_n\} ¤_k \{P_1,\ldots, P_m\} = \{ S_i ¤_k P_j \mid 0<i \leq n \text{ et } 0<j \leq m\}$$

The k-telescoped concatenation of *n* sequences $S_1, S_2, \ldots, S_n$ is:

$$Op¤_k(S_1, S_2,\ldots, S_n) = Op¤_k(S_1, Op¤_k(S_2,\ldots, S_n))$$

E.g., $Op¤_3(ACDE,CDEF,DEFD) = Op¤_3(ACDE,Op¤_3(CDEF, DEFD)) = Op¤_3(ABCD,CDEFD) = ACDEFD$.

The union of *n* sequences $S_1, S_2, \ldots S_n$ is the set of sequences:

$$Op+(S_1, S_2,\ldots, S_n) = \{S_1, S_2, \ldots, S_n)\}$$

The union of two sets of sequences is the union of these sets:

$$\{S_1,\ldots, S_n\} + \{P_1,\ldots, P_m\} = \{S_1,\ldots, S_n, P_1,\ldots, P_m\}$$

The *Kleene closure* applies to a set of sequences and denotes all the sequences one can build from them using concatenations. It includes the empty set **Ø**.

$$Op*\{S_1,\ldots, S_n\} = \{ \mathbf{Ø}, \{S_1, \ldots, S_n\} ¤_k \{S_1, \ldots, S_n\}, Op*(\{S_1,\ldots, S_n\} ¤_k \{S_1,\ldots, S_n\}, S_1,\ldots, S_n)\}$$

Moreover, the function *frequent* applied to a set of sequences $\{S_1,\ldots, S_n\}$ scans the database and returns a set that contains the frequent sequences.

The operators have a variable arity. The priority increases from + to $¤_k$ and from $¤_k$ to *. The concatenation can be distributed over the union. The union and the concatenation are associative. When all the possible concatenations have taken place, the resulting sequence is called an *atomic sequence*. Consider the RE-constraint B.CD.E.A(H+F) which can be transformed into BCDEA(H+F) by 3 concatenations. According to our definition, BCDEA is a newly formed atomic

sequence. H and F were already atomic sequences as they cannot be concatenated to their neighbors, but B, CD, E and A are not as they can be packed together to form a longer sequence. The building bricks of our RE-constraints are the atomic sequences, i.e., the smallest elements considered during the extraction phase.

Canonical form We say that a regular expression is in a *canonical form* if it contains only atomic sequences. In the following, we assume that all the REs are in the canonical form.

Examples of RE-constraints and their associated derivation phrases:
A+BE+CF+D = Op+(A, BE, CF, D)
A(B)*(CF+D) = Op$¤_0$(A, Op*(B), Op+(CF,D))

Sub-constraints are taken from the initial RE-constraint: they correspond to terms of the derivation phrase. Extraction must not break the priorities of the operators. E.g., B+C can not be extracted from A$¤_0$B+C as the priority of the concatenation prevails over the union. Besides, a sub-constraint must contain as many terms as the arity of the operator in the initial constraint. A *maximal sub-constraint* is a sub-constraint, which is not contained in any other sub-constraint except the initial constraint. E.g., A$¤_0$B+C has two maximal sub-constraints: A$¤_0$B and C. The maximal sub-constraints naturally define partitions over the initial RE. The *active operator* connects the maximal sub-constraints of a given constraint. E.g., the active operator for A$¤_0$B+C is the union.

Hackle-tree. A *Hackle-tree* is an AST which encodes the structure of the canonical form of a RE-constraint. Every inner node of this tree corresponds to an operator, and the leaves contain atomic sequences of (possibly) unequal lengths. The tree reflects the way in which these atomic sequences are assembled by the operators to form the initial RE-constraint. Figure 1 provides such a tree for the RE-constraint C((C(A+BC)D)+(A+B+C)*)C. Nodes are marked with roman numbers to support the discussion. Attributes associated to each node have been described in [2] and are given in the following table:

Attribute	Semantics
type	Type of the node: $\perp$ leaf, ¤ concatenation, + union, * Kleene closure.
siblings	List of the siblings. NULL for the leaves.
parent	Parent of the node. NULL for the root.
ξ_{th} ,ξ_{exp}	Theoretical and experimental cardinality of the node.
items	Frequent legal sequences found by the node.
state	*Unknown* – The exploration of the node has not yet begun. *Satisfied* – Exploration has found frequent legal sequences. *Violated* – The node did not generated a frequent sequence.
explored	Coupled to the attribute State. True if the exploration is completed.
K	Parameter for the k-telescoped concatenation: $¤_k$
age	Only for Kleene closures: counts the times the node has been visited.
seq	Only for the leaves: encoded atomic sequence or symbol.

The construction of the tree is not trivial. The RE has to be transformed in its canonical form and arities have to be computed from the number of the maximal sub-constraints. Then a node is created for the active operator as well as a sibling

for each maximal sub-constraint, which is expanded recursively. This method minimizes the height of the tree and can be encoded efficiently.

The intuitive notion of selectivity can be formalized using cardinalities [2]. The *theoretical cardinality* of a constraint is the number of sequences it can generate after the expansion of all the operators if every possible sequence is in the database. The *experimental cardinality* of a constraint is the number of sequences extracted from the database. While the theoretical cardinality refers only to the constraint, the experimental cardinality takes into account the database instance, i.e., the results of the counting phases.

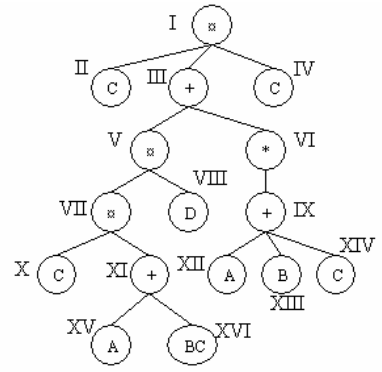


Figure 1 *Hackle-tree encoding constraint* C((C(A+BC)D)+(A+B+C)*)C.

The *extraction phrase* Ψ is the list of the nodes one must examine at a given step. The whole extraction process is controlled by this phrase: its modification defines a new generation and thus a new database scan. When the extraction starts, Ψ contains all the leaves of the tree collected from left to right. It is updated after each database scan by replacing the explored nodes with their parents.

The *extractor functions* (denoted C) are applied to the nodes of the Hackle-tree, and return the candidates that have to be counted. They are defined as follows:

- Leaves: C(N) = N.seq
- Concatenation: all siblings of N must be explored,
  C(N) = ¤$_k$ M.items, for all M∈N.siblings , k comes from the node N
- Union: all siblings of N must be explored,
  C(N) = ∪ M.items, for all M∈N.siblings
- Kleene closure: the sub-tree of N must be explored,

$$C(N) = \bigcup_{age>0} C(N, age)$$

with C(N,age+1)=¤$_{age-1}$frequent{C(N,age)}and C(N,1)=N.siblings.items

3. The RE-Hackle Algorithm

RE-Hackle extracts all the frequent sequences which match a given regular expression, i.e., which are valid w.r.t. the root of its Hackle-tree. Details about the basic algorithm are given in [2]. Here, we just provide informal comments that enable to introduce our new optimization and the extension to RE-constraints with variables. The algorithmic schema is as follows (T is a Hackle-tree, E a RE-constraint, Ψ the extraction phrase, and C the set of candidates):

```
T ← BuildTree (E)
loop    Ψ ← BuildExtractionPhrase (T)
        C ← GenerateCandidates (Ψ)
        CountCandidates (C)
        T ← TransformTree (T)
while Ψ<>Ø and C<>Ø
return T.root.items
```

Extraction relies on the extraction phrase. It starts at the leaves of the Hackle-tree and lasts several generations. At every generation, the extraction functions are applied to the nodes in the extraction phrase; the algorithm counts the candidates and uses the frequent ones to feed the next generation (an upper level) which will be assembled by the new nodes of the updated extraction phrase. It is a levelwise algorithm in the structured space of the language associated to the RE-constraint. The number of the levels is limited by the height of the Hackle-tree plus the number of times Kleene nodes are evaluated. As candidates are built up from atomic sequences, it takes usually less database scans than GSP-like algorithms [1]. The Hackle-tree is transformed after each generation, e.g., for pruning branches which can no longer generate new candidates.

Let us comment an execution of the algorithm given the following database, a minimal frequency of 2, and the RE-constraint from Figure 1.

ID	Sequences
1	CCADCABC
2	ECBDACC
3	ACCBACFBAC
4	CCBAC

The extraction needs for 7 generations and 6 database scans.

1st Generation

Ψ_1 = II,X,XV,XVI,VIII,XII,XIII,XIV,IV
Candidates: A, B, C, D, BC
Frequent sequences: A, B, C, D

BC is not frequent and the algorithm prunes Node XVI.

2nd Generation

Ψ_2 = VI,VII
Candidates: AA, AB, AC, BA, BB, BC, CA, CB, CC
Frequent sequences: AB, AC, BA, CB, CC

Ψ_2'= XI,IX would have been the application of the defined principle: the substitution of Nodes XV and XVI by XI and the substitution of Nodes XII, XIII and XIV by IX. In fact, as unions never need to access the database, the algorithm replaces them with their nearest concatenation or Kleene closure parent during an intermediate generation. It is not necessary to compute explicitly Ψ_2'. While processing Ψ_2, no frequent sequence has been found at node VII, so it is erased from the tree. Its parent, the node V, can be pruned too thanks to the anti-monotonicity of the minimal frequency constraint.

3rd Generation

Ψ_3 = VI *(age=1)*
Candidates: ABA, ACB, ACC, BAB, BAC, CBA, CCB, CCC
Frequent sequence: ACC, BAC, CBA, CCB

A Kleene Closure node remains in the extraction phrase while it continues to generate frequent sequences.

4th Generation

Ψ_4 = VI *(age=2)*
Candidates: ACCB, BACC, CBAC, CCBA
Frequent sequences: CBAC, CCBA

ACC ¤$_2$ CCB = ACB. Parameter k of the concatenation is given by the age of the Kleene node.

5th Generation

Ψ_5 = VI *(age=3)*
Candidates: CCBAC Frequent sequences: CCBAC

6th Generation

Ψ_6 = VI *(age=4)*
Candidates: - Frequent sequences: -

No candidate to count since CCBAC ¤$_4$ CCBAC = ε.

7th Generation

Ψ_7 = I
Candidates: CC, CAC, CBC, CCC, CABC, CACC, CBAC, CCAC, CCBC, CCCC, CACCC, CBACC, CCBAC, CCCBAC, CCBACC, CCCBACC
Frequent sequences: CC, CBAC, CCBAC

The Kleene closure returns every frequent combinations of A and B, plus the empty sequence. The root assembles them to C and generates the result, i.e., the frequent items associated to the root. Notice that the candidates of the k^{th} generation are not necessarily of length k.

Let us now discuss the processing of the Kleene closure nodes thanks to *second-level alphabets*. For each atomic sequence of a constraint, we are defining a new symbol which replaces it in an equivalent second-level RE-constraint. For example, the RE-constraint A(AB|FE)DCA becomes A(α|β)χ given the new symbols α=AB, β=FE and γ=DCA. Every initial symbol such as A is added automatically to the second-level alphabet. The RE-Hackle algorithm works with second-level alphabets.

The nodes corresponding to the Kleene closures collect the frequent sequences extracted by their descendents. These sequences of unequal length must be combined at later ages to compute the closures.
By definition, the *age* of a Kleene closure is the number of times it has been visited (it begins with 1). Assume that A, ADC and BAD have been found frequent by the siblings of a Kleene node, it means that the closure of {A,ADC,BAD} must be computed. The candidates should be {AA, AADC, ABAD, ADCA, ADCADC, ADCBAD, BADA, BADADC, BADBAD}. Assume now that all of them are frequent. At the third age, the node must concatenate only the sequences, which share a common sub-sequence of the first generation, i.e., A, ADC or BAD. For example, even though ABAD and ADCA share a common sequence of length 2, they should not be concatenated because ABADC does not belong to {A,ADC,BAD}*. With a representation that keeps no information about the composition of the second (n^{th}) age sequences, the overlapped concatenation of these sequences is practically impossible. It has motivated the introduction of second-level alphabets. Dealing with {A,β,γ} given that β=ADC and χ=BAD is an elegant solution for candidate generation. The second level candidate Aβ will be converted to its first level representation, i.e., AADC for counting purposes and handled in its second level representation when computing overlapped concatenations. Notice that in our hierarchical representation, the overlapping parts of the sequences are easily identified. For instance, the sequence Aβχ = (A)(ADC)(BAD) contains only three symbols in its second level representation, which is a shortcut for a flattened sequence of length 7. Candidates are generated in the same way as with GSP [1] where the age of the Kleene node encodes the number of the overlapping second-generation sequences (parameter k for the k-telescoped concatenation $¤_k$). The use of a second level alphabet can boost the extraction of the Kleene closure, as the candidates are assembled from sequences rather than from simple symbols. Consequently, their length grows faster then with any previous method and the number of database scans can drop drastically.

4. Optimization

Let us introduce an original optimization for RE-Hackle, the so-called *ascending flux of candidates*. It can be easily shown that the exploration of each Kleene node delays by one generation the evaluation of its parent, as parents can not be put in the extraction phrases if the extraction of their siblings is not completed. Although generally it does not penalize the extraction (the algorithm can continue to work with some other branches of the tree), it would be better to have a guarantee that the embedded structures will never lack of efficiency. Therefore, we decided to forward the frequent sequences found by a Kleene node immediately to its parent, i.e., without waiting for the completion of the exploration. Indeed, the extraction against constraint A*B can be inefficient if the database contains long sequences of symbols A but does not contain AB. We can forward the extracted sequences immediately after having found them. Thus, the optimized algorithm can work on several levels of the Hackle-tree. The extraction phrase is now allowed to contain nodes which are in direct son-parent relations.

The reconstruction of the extraction phase becomes more complex, so does the propagation of the violations and the candidate generation. Technically, it is achieved by 3 possible values for attribute *explored*:

- Waiting: the exploration of the node has not yet begun
- Inprogress: the exploration has begun, but the node will generate some more candidates during the next generation
- Finished: the exploration is finished and the node has been taken out from the extraction phrase.

If the parent of a closure is a concatenation which generates no frequent sequence at a given iteration, then its sub-trees can be pruned immediately and the exploration of the closure can be stopped. The candidates from the n^{th} age can be counted together with the frequent sequences of the $(n-1)^{th}$ age concatenated to the other siblings of the concatenation. In this way, when the n^{th} age becomes empty, the concatenations of the $(n-1)^{th}$ age with the other siblings are available and no iteration is wasted. In Table 1, we compare the basic and the optimized algorithm when AAAACBB is mined given the RE A*B. The optimized version recognizes that AB does not occur in the database and it does not generate useless sequences of A. Here, the optimization saves four database scans.

Scan	RE-Hackle (basic)	RE-Hackle (optimized)
1	A, B	A, B
2	AA,B	AA, B, ~~AB~~
3	AAA	STOP
4	AAAA	
5	~~AAAAA~~	
6	~~AB~~, ~~AAB~~, ~~AAAB~~,~~AAAAB~~	
7	STOP	

Table 1 *Ascending flux of candidates. Infrequent candidates are stroked.*

In some cases, the basic algorithm can give rise to a large number of candidates without any pruning. For instance, assume a concatenation with a dozen of siblings each of them producing 3 to 4 frequent sequences. The extractor functions would return between 531.441 to 16.777.216 candidates and this is clearly unacceptable.

Fortunately, the *adaptation of the extraction method* can avoid this combinatorial explosion. It is always possible to group the nodes in larger overlapping buckets, and to benefit of more frequency-based pruning. Figure 2 illustrates this principle. Assume that the siblings A, B, C and D of the node X return many frequent sequences (*Here, A,B,C, and D denote the nodes of the tree and are not elements of the alphabet*), and that we do not want the concatenation to produce a large number of candidates. So, to protect X, we replace it by a new node X^2 which introduces a new level (nodes Y, Z and W). The initial nodes are grouped two by two and associated to Y, Z and W to enable pruning. As the suffixes of the sequences in Y are the same as the prefixes of Z, the candidates of X^2 will be constituted by a *1-telescoped concatenation*.

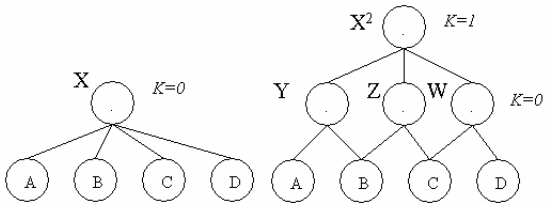


Figure 2 *– Adaptation of the extraction method.*

After the evaluation of the additional level the number of the candidates should globally decrease. So, X^2 contains only three sons and it is supposed to generate fewer candidates than X. If this number is still too large, the algorithm can decide to introduce one more additional level between Y, Z, W and X^2. The number of the inserted levels is determined by the algorithm during the extraction by the use of the theoretical cardinality. If the theoretical cardinality of a concatenation node exceeds a level, this optimization technique introduces a new level between the node and its siblings. Each new level requires another whole database scan, but the number of the candidates is expected to decrease. This mechanism enables a tradeoff between the number of the candidates and the number of database scans. One should notice, that the length of the candidates increases with each level, so the RE-Hackle will never take more passes than GSP or SPIRIT(L and V). The control of the tradeoff is however complex: it depends on the size of the database, the cost per candidate for counting and the cardinalities of the siblings of the node we are considering.

5. Experimental Results

We have used a semi-optimized implementation of the RE-hackle algorithm for our experiments (no ascending flux of candidates). Furthermore, we have put the data in main memory for both our SPIRIT and RE-Hackle implementations. We used our fair implementations of SPIRIT(L) and SPIRIT(V) [3]. First, we have generated synthetic datasets following a zipfian distribution. First, our synthetic dataset contains 100k transactions of length 20 over an alphabet of 100 symbols. We have then decreased the number of the symbols in the alphabet and created conditions for the emergence of long patterns. The execution time of our algorithm (see Figure 2) does not increase exponentially and take the properties of the better algorithm for this case, i.e., SPIRIT(V).

We have been considering different RE-constraints with different granularities (ratio between the number of symbols composing the RE and its number of atomic sequences). E.g., the granularity of AB(CDE|RY)CF is 9/4=2.25, for AC|T(G|BB)E we have 7/5=1.4, and for (A|B|C)*D(T|Z) we get 6/6=1. The execution time of RE-Hackle decreases as the granularity increases (see Figure 4). Notice, that the SPIRIT algorithms are practically insensible to this factor.

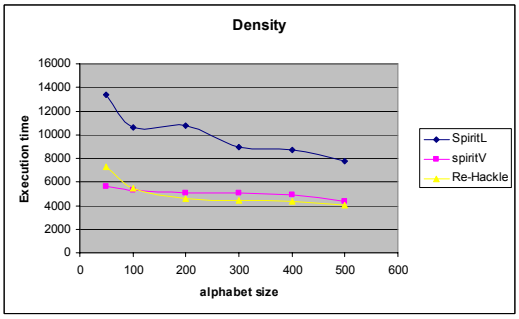


Figure 3 *Influence of the data density*

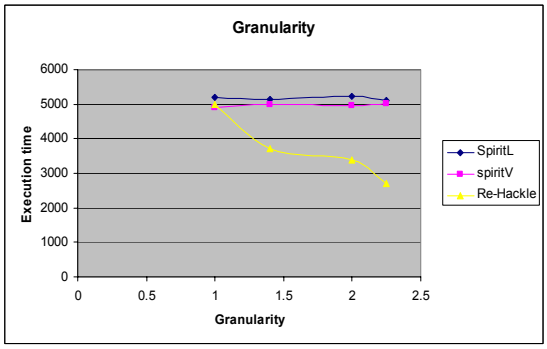


Figure 4 *Influence of RE granularity*

The power of RE-Hackle pruning strategy is somewhat between SPIRIT(L) and SPIRIT(R)[2]. As SPIRIT(L) relies on frequency-based pruning and SPIRIT(R) on RE-based pruning, RE-Hackle will not always beat both of them, but it is quite often the best, as it uses both pruning criteria. We provide in Figure 5 a comparison between RE-Hackle, SPIRIT(L) and SPIRIT(V) on real biological data (5000 sequences of length 3000 to 6000 items over an alphabet of four symbols given by Dr. O. Gandrillon from University Lyon 1). Various RE-constraints (with an increasing experimental cardinality) have been used. As the pruning strategy is adjusted dynamically according to the cardinality of the constraint and the content of the database, RE-Hackle approaches the performances of the best SPIRIT implementation without prior knowledge of the selectivity of the constraint. This behavior was mainly obtained by the use of the adaptive extraction method.

[2] [3] introduces 4 SPIRIT algorithms denoted N, L, V and R. Following this order, the algorithms do more and more RE-based pruning (almost no RE-pruning with SPIRIT(N), no frequency-based pruning for SPIRIT(R)).

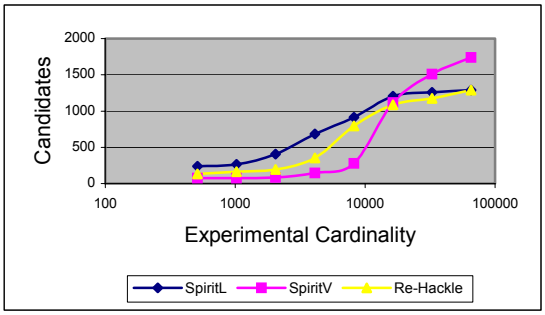


Figure 5 *Comparison to SPIRIT variants*

This is an important qualitative result. It is not surprising that both SPIRIT algorithms have their operating areas and RE-Hackle is situated between the two curves. As the experimental cardinality rises, RE-Hackle begins to use more and more frequency-based pruning and avoids the combinatorial explosion which makes SPIRIT(V) inadequate to deal with high cardinality constraints. At a given complexity, RE-Hackle decides to use only frequency-based pruning and has the same behavior than SPIRIT(L). Indeed, we have an adaptive pruning method.

6. Extension to RE-constraints with variables

Our partners, the biologists, have shown interest in being able to specify variables in the RE constraints. For instance, it is interesting to look for frequent sequences that match a generalized RE like X(A|B)C*XCB (where X denotes a variable which take values in $\mathcal{A}^*$). Hackle-trees provide an elegant way for tackling variables which specify that recurring sub-patterns are desired.

A *variable* X[3] is a shortcut for any sequence built over $\mathcal{A}$. Starting from now, we use upper case letters for variables and sequences while lower case letters are used for symbols. An *extended RE-constraint* is a RE-constraint that can contain variables. For instance, constraint (A|C)XBX requires that every sequence begins with an A or a C and contains two occurrences of the same sub-sequence X separated by the symbol B. Sequences ACDBCD or CADBAD satisfy it. Sub-sequences CD and AD are two *instances* for the variable X.

An *augmented second-level alphabet* is a second-level alphabet that contains a new symbol for every distinct variable. An extended RE-constraint is built over an augmented second-level alphabet, i.e., an alphabet that contains some variables. Mining under extended RE-constraints is now straightforward using an X-tree, i.e., a Hackle-tree on the augmented second-level alphabet associated to the extended RE-constraint.

For practical reasons, we impose now that Kleene closures can not return the empty sequence. If the result of a closure is empty, the node is immediately violated and pruned out of the X-tree. As a variable is a shortcut for all possible frequent sequences, every distinct variable can be modeled by the most general Kleene closure, i.e., a closure applied to the initial alphabet A. A variable which appears only once can be

[3] We reserve letters X and Y to denote variables.

replaced immediately by a Kleene closure. We thus assume that a variable appears at least twice in the RE-constraint. It is important to instantiate the variables with values that have been generated in the same generation. In other terms, when the expressions with variables are flowing up in the tree, the so-called X-Hackle algorithm has to know from which generation they come from. Therefore, they can be indexed by the number of the active generation of the Kleene node. This count starts when they begin to flow up in the X-tree. As variables are implemented via Kleene closures, the corresponding nodes have the following fields.

Attributes	Semantics
type	* Kleene closure
siblings	All the symbols of the initial alphabet.
parent	Parent of the node. NULL for the root.
ξ_{th} , ξ_{exp}	Same as in a Hackle-tree.
state, explored	Same as in a Hackle-tree.
Age	Counts the times the node has been visited.
items[1..age]	Frequent legal sequences found by the node in generation

Figure 6 provides the X-tree representation of the generalized RE-constraint Y(AX*)BXY. It is not needed to represent explicitly Nodes 2 and 6: their parent points directly to Node 11. Also, Nodes 5 and 9 are shortcuts for Node 10.

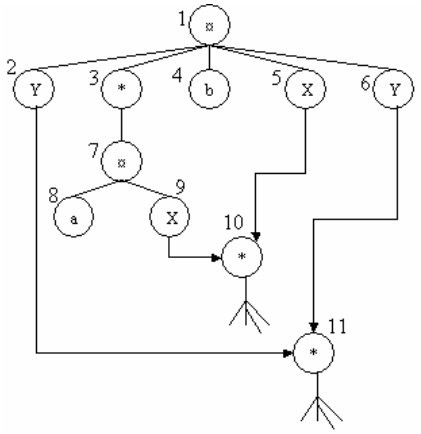


Figure 6 *X-Tree of the generalized RE-constraint* Y(AX*)BXY

Variables X and Y are shortcuts for two general Kleene closure nodes that will extract any possible frequent sequence. This setup combined with the ascending flux of candidates can extract the sequences described by generalized RE- expressions. Conceptually, the ascending flux of candidates can be viewed as the creation of a new X-tree at every generation even though, in practice, we can avoid the duplication of the tree. At the k^{th} evaluation of the variables, the algorithm creates a clone of the X-tree in which the k^{th} values of the variables are trapped. This tree is then explored by the RE-Hackle algorithm. At the next generation, X-Hackle creates a $(k+1)^{th}$ tree again and pass it to the RE-Hackle algorithm. Notice again, that in an implementation the

duplication of the tree can be avoided by clever indexing. We now sketch the X-Hackle algorithm that uses the following definitions.

Let us denote $First_k(N_1, ..., N_m)$ the function that, given a list of nodes, returns the set of the k-sequences (sequences of length k) which are prefixes of the concatenation of the nodes. Let us denote $Last_k(N_1, ..., N_m)$ the function that, given a list of nodes, returns the set of the k-sequences which are suffixes of the concatenation of the nodes.

The X-Hackle algorithm can now be sketched. A longer version of this paper contains more details and an example of execution.

(01)	`T ← BuildExpTree(E);`
(02)	`For all X∈Variables,` $V_1[X] = \emptyset$
(03)	`  For all A∈`$\mathcal{A}$
(04)	`  If Freq(A,D) > minsup`
(05)	`  Then` $V_1[X] \leftarrow V_1[X] \cup \{A\}$
(06)	`gen ← 1`
(07)	`While (Promising()) do`
(08)	`  ` $T_{gen} \leftarrow T$
(09)	`  ` $V_{gen}[.] \leftarrow$ `ComputeValues(`V_{gen-1}`[.])`
(10)	`  For k = 1 to gen`
(11)	`  Iterate RE-Hackle(`T_k`,` V_k`[.], k-gen+1)`
(12)	`  gen ← gen+1`
(13)	`Return` T_1`.root.items ∪ … ∪` T_{gen}`.root.items`

$V_{gen}[X]$ contains all the possible values for the X variable in T_{gen}. These are (gen)-sequences. ComputeValues($V_{gen-1}[.]$) computes the (gen-1)-concatenation of the (gen-1)-sequences for every variable X and returns the frequent (gen)-sequences. Iterate RE-Hackle(T_k, $V_k[.]$, c) performs an RE-Hackle iteration on the T_k tree using the values from $V_k[.]$ for the instanciation of the variables. If T_k is already explored it stops. The third parameter c corresponds to RE-Hackle generation for the k^{th} tree. Promising() is a boolean function which decides wether X-Hackle should continue or not. Just to give a hint, it takes all the concatenations containing variables and computes all the possible First and Last functions. E.g., during the evaluation of (XA)*B, if CAB is not frequent then it is useless to compute the other elements of the closure that finish by CA. The idea is very similar to the one used in the ascending flux of candidates optimisation technique. Formalisation will come in a future publication. If all the First and Last functions from a given node are violated (e.g., they don't generate any frequent sequence) the node is pruned. The algorithm finishes when the X-tree gets empty.

7. Conclusion

We have proposed a framework that characterizes and exploits the local properties of RE-constraints to benefit of both RE-based and frequency-based pruning. Our solution computes dynamically a balance between the two pruning strategies and restricts the search space for arbitrary RE-constraints. It enables to take the shape of the best SPIRIT ad-hoc algorithm without any prior knowledge of the selectivity of the constraint. Thanks to the two-level alphabets, complex expressions are handled efficiently. The cardinalities of a sub-expression appear useful for choosing the

extraction method. It enables to introduce global information such as database access costs in the reorganization of the extraction structures. This adaptative strategy associated to powerful optimization techniques such as the transformation of Hackle-trees introduced in [2] or the ascending flux of candidates introduced here enable to tackle RE-constraints even though they are neither anti-monotonic nor monotonic constraints on the search space of the initial symbols. The RE-Hackle approach opens a new framework for the extraction of frequent sequences that satisfy rich syntactic constraints. We did not yet identify all the possible uses of this hierarchical constraint. However, we suspect that a larger (w.r.t. the class of RE-constraints) type of constraint can be handled within this framework. We started to consider RE-constraints with variables. Good properties of the framework are preserved and the potential for applications is clearly enlarged since RE-constraints with variables enable to express context-sensitive restrictions. Our future work concerns further optimizations for the RE-Hackle algorithm. Furthermore, we have to implement X-Hackle and look for a relaxation of the pruning strategy, as the computation of the Last/First sets is expensive. We would like to find a faster termination condition as well.

Acknowledgements. This research is partially funded by Région Rhône-Alpes (programme d'appui à la société Djingle) and the Future and Emerging Technologies arm of the IST programme (project cInQ IST-2000-26469).

References

[1] R. Agrawal, R. Srikant. Mining sequential patterns. Proceedings *ICDE'95, Tapei (Taiwan), 1995. pp. 3-14.*

[2] H. Albert-Lorincz, J-F. Boulicaut. Mining frequent sequential patterns under regular expressions: a highly adaptive strategy for pushing constraints (poster paper). Proceedings *SIAM DM'03, San Francisco (USA), May 1-3, 2003. pp. 316-320.*

[3] M. Garofalakis, R. Rastogi, K. Shim. SPIRIT: Sequential Pattern Mining with Regular Expression Constraints. Proceedings *VLDB'99, Edinburgh (UK), 1999. pp. 223-234.*

[4] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, M.-C. Hsu. FreeSpan: Frequent Pattern-Projected Sequential Pattern Mining. Proceedings *SIGKDD'00, Boston (USA), 2000. pp. 355-359.*

[5] H. Mannila, H. Toivonen. Levelwise search and borders of theories for knowledge discovery. *Data Mining and Knowledge Discovery journal,* Vol. 1(3),1997, pp. 241-258.

[6] H. Mannila, H. Toivonen, and A. I. Verkamo. Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery journal,* Vol. 1(3),1997, pp. 259-289.

[7] M. Leleu, C. Rigotti, J-F. Boulicaut, G. Euvrard. Constraint-based sequential pattern mining over datasets with consecutive repetitions. Proceedings *PKDD'03, Catvat-Dubrovnik (Croatia), 2003.* To appear.

[8] J. Pei et al. PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth. *Proceedings ICDE'01, Heidelberg (D), 2001. pp. 215-224.*

[9] J. Pei, J. Han, and W. Wang. Mining sequential patterns with constraints in large databases. *Proceedings CIKM'02, McLean (USA), 2002. pp. 18-25.*

[10] R. Srikant, R. Agrawal - Mining Sequential Patterns: Generalizations and perfor-mance Improvements. *Proceedings EDBT'96, Avignon (F), 1996. pp. 3-17.*

[11] M. J. Zaki. SPADE: An Efficient Algorithm for Mining Frequent Sequences. *Machine Learning Journal, Vol. 42(1/2*), 2001, pp *31-60.*

[12] M. J. Zaki. Sequence Mining in Categorical Domains: Incorporating Constraints. *Proceedings CIKM'00, Washington (USA), 2000, pp. 422-429.*

MR-SMOTI: A Data Mining System for Regression Tasks Tightly-Coupled with a Relational Database

Annalisa Appice Michelangelo Ceci Donato Malerba

Dipartimento di Informatica, Università degli Studi
via Orabona, 4 - 70126 Bari - Italy
{appice, ceci, malerba}@di.uniba.it

Abstract. Tight coupling of data mining and database systems is a key issue in inductive databases. It ensures scalability, direct and uniform access to both data and patterns stored in databases, as well as proper exploitation of information embedded in the database schema to drive the mining process. In this paper we present a new data mining system, named Mr-SMOTI, which is able to mine (multi-)relational model trees from a tightly coupled relational database. The induced model tree is a (multi-)relational pattern that can be represented by means of a set of selection graphs, which are translated into SQL expressions and stored in XML format. A peculiarity of induced model trees is that they can represent both local and global effects of variables used in regression models. This distinction between local patterns and global models addresses a limitation of current inductive database perspective, which mainly focus on local pattern mining tasks. Preliminary experiments demonstrate the ability of Mr-SMOTI to mine accurate regression predictors from data stored in multiple tables of a relational database.

1 Introduction

The integration of data mining with database systems is an important issue in inductive database research. Most data mining systems still process data in main memory. This results in high performance for computationally intensive processes when enough memory is available to store all necessary data. However, a common aspect of many data mining algorithms is their frequent access to data that satisfy some selection conditions. For data intensive processes, it is important to exploit powerful mechanisms for accessing, filtering and indexing data, such as those available in database management systems (DBMS). This motivates a tight coupling between data mining and database systems. In an inductive database perspective, this tight coupling also aims to support a direct and uniform access to both data and patterns stored in databases. Other equally important reasons are: i) the applicability of data mining algorithms to large data sets; ii) the exploitation useful knowledge of data model available, free of charge, in the database schema, iii) the possibility to specify directly what data stored in a database have to be mined, without any pre-processing.

The last reason is even more justified by the emergent trend in KDD research, namely (multi-)relational data mining [8], which looks for patterns that involve *multiple* relations of a relational database. Thus data taken as input by relational data mining systems typically consists of several tables and not just a single one. The *single-table assumption* [28] forces the user of traditional data mining systems to perform complex SQL queries in order to compute a single table whose rows (or tuples) represent independ ent units of analysis.

Some examples of integration of data mining and database systems are presented in [24] for association rules, in [18] for clustering and in [25] for decision trees. In [18] a system named MiningMart has been proposed for approaching the knowledge discovery in database by building upon database facilities and integrating data mining algorithms into the database environment. In all these works it has also been advocated the importance of implementing some data mining primitives to implement them using DBMS extension facilities, e.g. packages, cartridges, extenders or datablades. In [1] a package implemented in Oracle Spatial has been presented to support the extraction of spatial relations between geographical objects. This is also a rare example of (multi-)relational data mining system, named SPADA, (loosely) integrated with an object-relational spatial database. Other two examples of tight integration of (multi-)relational data mining systems with a database are MRDTL [14] and SubgroupMiner [11]. These three examples refer to the tasks of association rule mining, classification (with decision trees), and subgroup discovery, respectively.

In this work, we present Mr-SMOTI, a prototypical example of multi-relational data mining system for *regression* tasks that is tightly integrated with a relational database, namely Oracle^{R} 9i. Differently from traditional data mining regression systems (e.g. M5 [22], RETIS, [9], M5' [27], HTL [26], TSIR [15], SMOTI[16]) Mr-SMOTI directly works on complex and structured objects represented through *multiple* tables, and discovers *relational regression models* that involve attributes of several tables related by foreign key constraints.

The idea of mining regression models from data distributed in multiple tables is not new. The problem is generally solved by *moulding* a relational database into a single table format, such that traditional attribute-value algorithms are able to work on [6]. In contrast, relational regression models can be induced by formulating the problem in the *normal* ILP framework [5], where multiple relations can be directly managed through first-order representations. FORS [10], SRT [13], S-CART [8] and TILDE-RT [2] are examples of systems that solve relational regression problems by working on data stored as Prolog facts. This means that a little attention has been given to data stored in relational database and to how knowledge of data model can help to guide the search process.

Contrarily to previous works, Mr-SMOTI directly deals with multiple tables or relations as they are found in today's relational databases. Induced relational model trees can contain both regression nodes, which perform only straight-line regression, and split nodes, which partition the feature space. The model associated to each leaf is then the composition of the straight-line regressions reported along the path from the root to the leaf. In this way, internal regression nodes contribute to the definition of multiple models and capture global effects, while straight-line regressions at leaves can only capture local effects. Global effect refers to the fact that the contribution of an attribute to a regression model can be reliably estimated on more training objects

than those associated to the leaf. This overcomes one limitation of the inductive database prospective proposed in [3] that addresses only local patterns mining tasks. Mr-SMOTI upgrades the propositional system SMOTI, which induces model trees from data stored in main memory in the form of a single table. Therefore, attributes involved in nodes of relational regression models induced by Mr-SMOTI can belong to different tables of the relational database. The join of these tables is dynamically determined on the basis of the database schema.

In the next section we draw on the multi-relational regression framework, based on an extended graphical language (*selection graph*), to mine relational model trees directly from relational databases, through SQL queries. In Section 3 we show how selection graphs can support the stepwise induction of multi-relational model trees from structural data. Some experimental results are reported in Section 4. Finally, we draw some conclusions and sketch possible directions of further research.

2 Regression problem in a multi-relational framework

Traditional research for a regression task in KDD has focused mainly on propositional techniques involving the attribute-value paradigm. This implies that relationships between fields of one tuple can be found, but not relationships between several tuples of one or more tables. It seems that this is an important limitation, since a relational database consists of a set of tables and a set of associations. Each association describes how records in one table relate to records in another table. Most associations correspond to *foreign key relations*. These relations can be seen as having two directions. One goes from a table where the attribute is primary key to a table where the attribute is foreign key (*one-to-many*), and the other one is in the reverse way (*many-to-one*). An object in a relational database can consist of several records fragmented across several tables and connected by associations (Fig. 1). Although the data model can consist of multiple tables, there must be only a single kind of object that is central to the analysis (*target table*). The assumption is that each record in the target table will correspond to a single object in the database. Any information pertaining to each object which is stored in other tables can be retrieved by following the associations in the data model. Once the target table has been selected, a particular numeric attribute of that table can be chosen for regression purposes (*target attribute*).

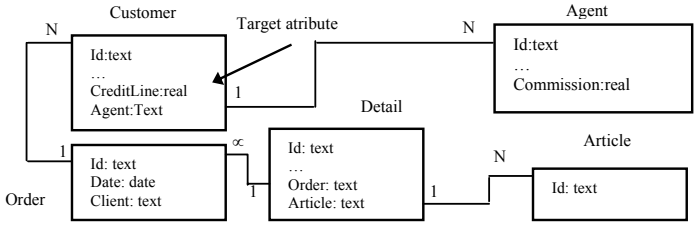


Fig. 1. The data model of an example database used in relational regression.

Thus, a multiple regression problem in a multi-relational framework can be defined as follows. Given a schema of a relational database D, a target table T_0, a target attribute Y within the target table T_0, the goal is to mine a multi-relational multiple regression model to predict the estimated target attribute Y. Mined models

not only involve attribute-value descriptions, but also structural information denoted by the associations in *D*.

Relational regression models, stepwise induced as in SMOTI can be expressed in the graphical language of *selection graphs*. The classical definition of a selection graph is reported in [12]. Nevertheless, we present an extension of this definition in order to make the selection graphs more appropriate to our task. In particular the selection graph must be able to represent a (multi-)relational regression model incrementally built. The incremental construction is based on the idea that when a new independent variable is added to the model its linear effect on remaining variables has to be removed [4].

Definition of *selection graph*

A selection graph G is a directed graph (N, A), such that:

– each node in N is a 4-tuple (T, C, R, s), named *selection node*, where:
  - $T = (X_1, X_2, \ldots X_n)$ is a table in the relational schema D.
  - C is a set of *conditions* on attributes in T of type $T.X'_i \; OP \; c$, where X'_i is one of the attributes X_i in T after the removal of the effects of some variables already introduced in the relational regression model through regression nodes. *OP* is one of the usual comparison operators (<, ≥, in, not in …) and c is a constant value.
  - R is a set of tuples $R=\{(RX_j, \boldsymbol{\alpha_j}, \boldsymbol{\beta_j}) \mid j=1,\ldots,l\}$ where RX_j is a regression term already introduced in the multiple linear model, l is the number of such terms, $\boldsymbol{\alpha_j} = (\alpha_{j1}, \alpha_{j2}, \ldots, \alpha_{jn})$ and $\boldsymbol{\beta_j} = (\beta_{j1}, \beta_{j2}, \ldots, \beta_{jn})$ are the regression coefficients computed to remove the effect of each term RX_j from all numerical attributes in T:
    $X'_i = X_i - \sum_{j=1,\ldots,l} (\alpha_{ji} + \beta_{ji} \times RX_j) \quad \forall \; i = 1,\ldots,n$ and X_i is numerical
  - s is a flag with possible values *open* or *closed.*

– A, a set of tuples (p, q, fk, e), where:
  - p and q are selection nodes.
  - fk is a foreign key association between $p.T$ and $q.T$ in the relational schema D (one-to-many or many-to-one).
  - e is a flag with possible values *present* or *absent*.

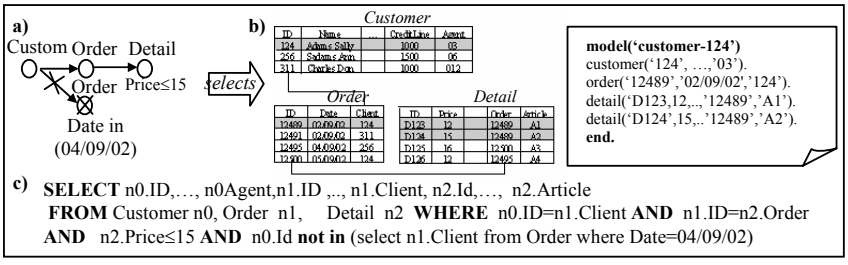


Fig. 2. (a) Example of selection graph; (b) corresponding grouping of data for an instance of the example database and (c) translation into an SQL query.

Selection graphs contain at least a node n_0 that corresponds to the target table T_0. They can be graphically represented by a directed labelled graph (Fig. 2.a). The value of s is expressed by the absence or presence of a cross in the node, representing the value *open* and *close* respectively. The value for e is indicated by the presence (*absent*

value) or absence (*present* value) of a cross on the corresponding arrow representing the labelled arc. The direction of the arrow (left-to-right and right-to-left) corresponds to the multiplicity of the association *fk* (one-to-many and many-to-one, respectively). Every arc between the nodes p and q imposes some constraints on how one or more records in the table $q.T$ are related to each record in table $p.T$ according to the list of conditions in $q.C$. The association between $p.T$ and $q.T$ induces some grouping (Fig. 2.b) in the records in $q.T$, and thus selects some records in $p.T$. In particular, a *present* arc selects those records that belong to the *join* between the tables and match the list of conditions. On the other hand, an *absent* arc corresponds to the negation of the joining condition and the representations of the complementary sets of objects. Intuitively, the tuples in the target table T_0 that are explained by a selection graph G are those for which tuples exist or not in linked tables that satisfy the conditions defined for those tables. The given definition of selection graph does not allow to represent recursive relationships. Therefore a selection graph can be straightforwardly translated into either SQL or into first order logic expressions (Fig. 2.c). In this case a subgraph pointed by an absent arc is translated into a negated inner sub-query.

3 Multi-relational stepwise model tree induction

Mr-SMOTI induces model trees whose nodes (regression, split or leaf) involve multi-relational patterns that can be represented with *selection graphs*, that is *each node of the tree corresponds to a selection graph*. Essentially Mr-SMOTI, like the propositional version SMOTI, builds a tree-structured multi-relational regression model by adding split and/or regression nodes through a process of successive refinements of the current selection graph until a stopping criterion is fulfilled and a leaf node is introduced. Thus, the model associated to each leaf is computed by combining all straight-line regressions in the regression refinements along the path from the root to the leaf.

3.1 The algorithm

Mr-SMOTI is basically a *divide-and-conquer* algorithm that starts with a root selection graph G containing only the target node n_0. This graph corresponds to the entire set of objects of interest in the relational database D (the target table T_0). At each step the system chooses the optimal refinement (split or regression) according to a heuristic function. In particular, a split refinement corresponds to either the updating of an existing node by adding a new selection condition or the introduction of a linked node in the current selection graph. On the other hand, a regression refinement corresponds to update the list of regression terms in existing nodes in order to remove the linear effect of those numeric attributes already included in the model Thus, descendants of a regression node must operate on modified training data. This transformation is coherent with the statistical procedure for the incremental construction of multiple linear regression models, according to which each time a new independent variable is added to the model its linear effect on remaining variables has to be removed [4].

The eventually modified training tuples selected by the optimal refinement (and its complement in case of a split), are used to select the regression functions

associated to the root of the left (/right) branch. This procedure is recursively applied to each branch until a stopping criterion is fulfilled.

Mr-SMOTI (D: database, G: selection_graph)
G_S, G_R, R: selection_graph; T_left, T_right: model_tree;
G_R:= *optimal_regression_refinement* (G, D);
if *stopping_criteria* (G_R, D) then return *leaf* (G_R);
G_S:= *optimal_split_refinement* (G, D);
R:= *best_refinement* (G_R, G_S);
if(R=G_R) T_left := *Mr-SMOTI* (D,R); T_right := ∅;
else T_left := *Mr-SMOTI* (D,R); T_right := *Mr-SMOTI* (D, *comp* (R));
return *model_tree*(R, T_left, T_right).

The functions *optimal_split_refinement* and *optimal_regression_refinement* take the selection graph *G* associated to the current node and consider every possible split and regression refinement. The choice of which refinements are candidates is determined by the current selection graph *G*, the structure of data model in *D*, and notably by the multiplicity of associations within this data model. The validity of either a split refinement (G_S) together with its complement (*comp*(G_S)), or a regression refinement (G_R) is based on two distinct evaluation measures, $\sigma(G_S, comp(G_S))$ and $\rho(G_R)$, respectively. Let *T* be the multi-relational model tree currently stepwise built, *G* the selection graph associated to the node *t* in *T* and t_{Gs} ($t_{comp(Gs)}$) the left (right) child of *t*, associated to a split refinement G_S (the complementary split refinement *comp*(G_S)) of the selection graph *G*, $\sigma(G_S, comp(G_S))$ is defined as:

$$\sigma(G_S, comp(G_S)) = \frac{N(t_{G_S})}{N(t_{G_S}) + N(t_{comp(G_S)})} R(G_S) + \frac{N(t_{comp(G_S)})}{N(t_{G_S}) + N(t_{comp(G_S)})} R(comp(G_S)),$$

where $N(t_{Gs})$ ($N(t_{comp(Gs)})$) is the number of training tuples covered by the refinement G_S (*comp*(G_S)), and $R(G_S)$ ($R(comp(G_S))$ is the resubstitution error of the left (right) child, computed as follows:

$$R(G_S) = \sqrt{\frac{1}{N(t_{G_S})} \sum_{j=1}^{N(t_{G_S})} (y_j - \hat{y}_j)^2} \qquad R(comp(G_S)) = \sqrt{\frac{1}{N(t_{comp(G_S)})} \sum_{j=1}^{N(t_{Comp(G_S)})} (y_j - \hat{y}_j)^2}.$$

Therefore the evaluation measure $\sigma(G_s, comp(G_s))$ is coherently defined on the basis of the partially defined multiple linear regression models $\hat{Y}$ built by combining the best straight-line regression associated to t_{Gs} ($t_{comp(Gs)}$), with all regressions introduced along the path from the root to t_{Gs} ($t_{comp(Gs)}$).

In the case of a regression refinement G_R, the definition of a heuristic evaluation function $\rho(G_R)$ of the effectiveness of G_R cannot be naïvely based on the resubstitution error $R(G_R)$ [16]. Indeed, the splitting test "looks-ahead" to the best multiple linear regressions after the current split is performed, while the regression step does not perform such a look-ahead. A fairer comparison would be to grow the model tree at a further level in order to base the computation of $\rho(G_R)$ on the best split refinement G_{Rs}, after the current regression refinement is performed. Therefore, $\rho(G_R)$ is defined as follows:

$$\rho(G_R) = min\ \{R(G_R),\ \sigma(G_{Rs}, comp(G_{Rs}))\}.$$

The function *stopping_criteria* determines whether the current optimal refinement must be transformed into a leaf according to the minimal number of *target objects*

(*minObject*) covered by the current selection graph and the minimal threshold for the *coefficient of determination* (*minR*) of the prediction function built stepwise [4].

The regression model built stepwise by Mr-SMOTI is a set of SQL queries, each of which is associated to a node in the tree. SQL queries are stored XML format and can be in turn the object of a query according to an inductive database perspective. Moreover, they can be applied to new instances stored in the relational database in order to predict an estimate of the unknown target attribute. The prediction is averaged by means of a grouping on the target objects.

3.2 The refinements

Split refinements are an extension of the refinement operations proposed in [12] to perform a split node in a multi-relational decision tree. Whenever a split is introduced in a model tree, Mr-SMOTI is in fact refining the selection graph associated to the current node, by adding either a condition or an open node linked by a present arc. Given a selection graph G, the *add condition* refinement returns the refined selection graph G_S by simply adding a split condition to an open node $n_i \in G.N$ without changing the structure of G.

The *add linked node* refinement instantiates an association of the data model D by means of a *present arc*, together with its corresponding table, represented as an *open node*, and adds these to the selection graph G. Knowledge of the nature and multiplicity is used to guide and optimise this search. Since the investigated associations are *foreign key associations*, the proposed refinements can have two directions: *backward* or *forward*. The former correspond to *many-to-one* associations, while the latter describe *one-to-many* associations in the data model. This means that a backward refinement of the selection graph G does not partition the set of target objects covered by G but extends their descriptions (training data) by considering tuples joined in the table which are represented by the new added node. Each split refinement G_s of type *add condition* or *add linked node* is introduced together with its complementary refinement ($comp(G_S)$) in order to satisfy the mutual exclusion principle. Let Q_G be the SQL or first order expression translating the selection graph G, and Q_{Gs} ($Q_{comp(Gs)}$) the expression translating the split refinement (complementary refinement) Q_{Gs} ($Q_{comp(Gs)}$). For each target object selected by Q_G exactly one of both queries (Q_{Gs} and $Q_{comp(Gs)}$) should succeed.

In [12], Knobbe *et al.* propose a complementary refinement named *add negative condition* that should solve the problem of mutual exclusion between an *add condition refinement* and its *complement*. If the node that is being refined does not represent the target table, $comp(G_S)$ must be built from G by introducing an absent arc from the parent of n_i to the clone of the entire sub-graph of G that is rooted in n_i. The introduced sub-graph has a root (a clone of the node to be refined) that is a closed node updated with the refinement condition that is not negated. In this way the query translating $comp(G_S)$ negates an entire inner sub-query and not simply a condition. As was observed in [14], this approach fails to build complementary refinements when the node to be refined is *not directly* connected to the target node. The example in Figure 4 proves that the proposed mechanism could build a refinement G_S (Fig 4.a) and a complementary refinement $comp(G_S)$ (Fig 4.b) that are not mutually exclusive. To overcome this problem the complementary refinement $comp(G_S)$ should be

obtained by adding an absent arc from the target node n_0 to the clone of the sub-graph containing the *entire join path* from the target node to the node to be refined. The introduced sub-graph has a root (a clone of n_0) that is a *closed* node and is updated with the refinement condition that is not negated. A new absent arc is also introduced between the target node and its closed clone. This arc is an instance of the implicitly relationship between the primary key of the target table and the own itself (Fig 4.c).

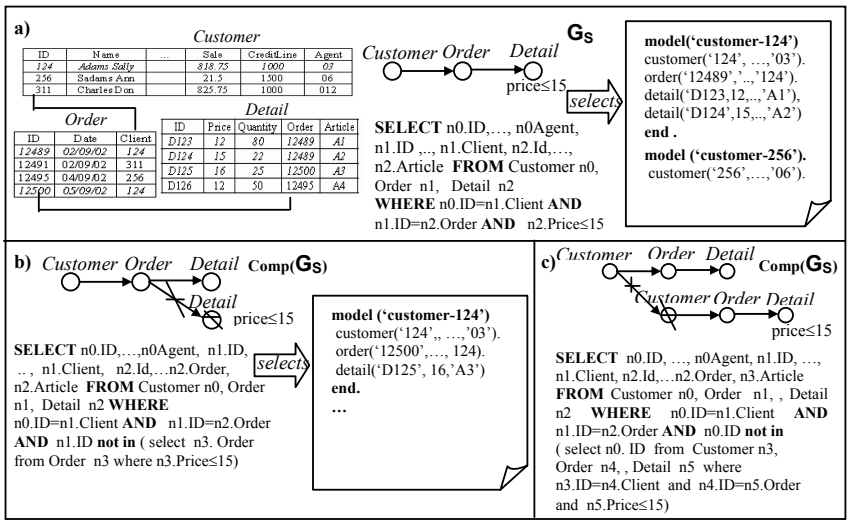


Fig. 4. Example of (a) refinement (G_S) by adding the a condition on a node not directly connected to the target node, (b) the corresponding complementary refinement, proposed in [11], that does not satisfy the mutual exclusion and (c) correct complementary refinement.

Similarly, when we consider the complementary refinement for an *add linked node* refinement we make the same considerations as when a negated condition is going to be added. This means that when the closed node to be added is not directly connected to the target node in G, a procedure similar to that described when an add condition refinement is complemented must be followed.

Finally, a *regression refinement* G_R of the selection graph G corresponds to performing a regression step ($Y'=\alpha_Y+\beta_Y \times n_i.T.X_j'$) on the residuals of a continuous attribute not yet introduced in the model currently. The coefficients α_Y and β_Y are estimated using all (joined) tuples selected by the current selection graph G. This means that the regression refinement is performed by considering a propositionalization of the (multi-) relational descriptions of the training objects selected by G. The regression attribute must belong to a table represented by a node in G. For each node, the list of regressions R is updated by adding the regression term ($n_i.T.X_j'$) introduced in the model and the coefficients α and β computed to update the residuals of all continuous attributes in the node.

4 Experimental evaluation

Mr-SMOTI has been applied to the biological problems of predicting both the mutagenic activity of molecules [19] and the biodegradability of chemical compounds in water [7]. *Mutagenesis* dataset consists of 230 molecules divided into two subsets: 188 molecules for which linear regression yields good results and 42 molecules that are regression-unfriendly. In our experiments we used the atom and bond structure of regression-friendly molecules by adding boolean indicators *Ind1* and *Ind2* as one setting (B_1) and adding *Lumo* and *Logp* properties to get a second setting (B_2). Similarly *Biodegradability* dataset consists of 328 chemical molecules structurally described in terms of atoms and bonds. In all the experimental results reported below the thresholds for stopping criteria are fixed as follows: *minObject*is is set to the square root of the number of target objects in the entire training set and *minR* must be below 0.80. Each dataset is analysed by means of a 10-*fold* cross-validation. Figure 5 shows the test set performance of *Mr-SMOTI* and *TILDE-RT* in both domains, as measured by the *Pearson correlation coefficient* that measures of how much the value of target attribute (y_j) in test objects correlates with the value predicted by the induced model. Since the Pearson correlation coefficient does not measure the quantity error of a prediction, we include several other measures [23] such the average error (*AE*) and the root mean square error (*RMSE*). For pairwise comparison with *TILDE-RT* the non-parametric Wilcoxon two-sample paired signed rank test is used [21]. The results of the Wilcoxon signed rank test on the accuracy of the induced multi-relational prediction model are reported in Table 1.

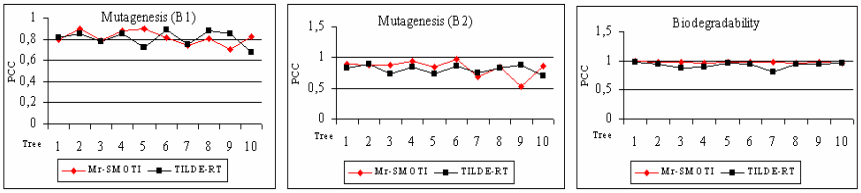


Fig. 5 Pearson correlation coefficient (Y axis) for multi-relational prediction models induced from the 10-fold cross validated datasets (X axis) of Mutagenesis (B1, B2) and Biodegradability datasets. The comparison concerns two systems: TILDE-RT (square) vs. Mr-SMOTI (diamonds).

Table 1. Results of the Wilcoxon signed rank test on the accuracy of the induced models. The best value is in boldface, while the statistically significant values ($p \leq \alpha/2, \alpha=0.05$) are in italics.

Dataset		*Accuracy*	*Mr-SMOTI*	*TILDE-RT*	*W+*	*W-*	*P*
Mutagenesis	*B1*	*Avg.MSE*	**1.165**	1.197	23	32	0.69
		Avg.AE	**0.887**	0.986	12	43	0.13
	B2	*Avg.MSE*	**1.118**	1.193	15	40	0.23
		Avg.AE	**0.845**	0.985	11	44	0.10
Biodegradability		*Avg.MSE*	***0.337***	0.588	0	55	*0.0019*
		Avg.AE	***0.186***	0.363	0	55	*0.0019*

The Wilcoxon test statistics W^+ (W^-) is the sum of the ranks from the positive (negative) differences between TILDE-RT and Mr-SMOTI. Therefore, the smaller

W^+ (W^-), the better for Mr-SMOTI (TILDE-RT). Differences are considered statistically significant when the p-value is less than or equal to $\alpha/2$. Interestingly all experimental results confirm the good performance of Mr-SMOTI.

5 Conclusions

This paper presents a novel approach to mining relational model trees. The proposed algorithm can work effectively when training data are stored in multiple tables of a relational DBMS. Information on the database schema is used to reduce the search space of patterns. Induced relational models are represented by selection graphs whose definition has been extended in order to describe model trees with either split nodes or regression nodes. As future work, we plan to extend the comparison of Mr-SMOTI to other multi-relational data mining systems on a larger set of benchmark datasets. Moreover, we intend to use SQL primitives and parallel database servers to speed up the stepwise construction of multi-relational model trees from data stored in large database. Finally, following the mainstream of our research on data mining query languages for spatial databases with an object-oriented logical model[17], we intend to pursue the investigation of defining a data mining query language appropriate to support both the discovery and the query of model trees.

Acknowledgments

This work has been supported by the annual Scientific Research Project "Metodi di apprendimento automatico e di data mining per sistemi di conoscenza basati sulla semantica" Year 2003, funded by the University of Bari. The authors thank Hendrik Blockeel for providing mutagenesis and biodegradability datasets.

References

[1]Appice A., Ceci M., Lanza A., Lisi F.A., Malerba D.: *Discovery of Spatial Association Rules in Georeferenced Census Data: A Relational Mining Approach*, Intelligent Data Analysis, numero speciale su "Mining Official Data" (in press).

[2]Blockeel H.:*Top-down induction of first order logical decision trees*. Ph.D thesis, Department of Computer Science, Katholieke UniversReit Leuven, 1998.

[3]De Raedt L.: A perspective on inductive databases. In *SIGKDD Explorations ACM,* Gehrke J (Ed.) Volume 4, Issue 2, 2002

[4]Draper N.R. & Smith H.: *Applied regression analysis*, John Wiley & Sons, 1982.

[5]Dzeroski S.: *Numerical Constraints and Learnability in Inductive Logic Programming*. Ph.D thesis, University of Ljubljana, Slovenia, 1995.

[6]Dzeroski S., Todoroski L. & Urbancic T: Handling real numbers in inductive logic programming: A step towards better behavioural clones. In *Machine Learning: ECML-95*, Eds. Lavrac N & Wrobel S., Springer , Berlin Heidelberg New York, 1995.

[7]Dzeroski S., Blockeel H., Kramer S., Kompare B., Pfahringer B., and Van Laer W.. Experiments in predicting biodegradability. *Proceedings of the Ninth International*

Workshop on Inductive Logic Programming (S. Dzeroski and P. Flach, eds.), LNAI, vol. 1634, Springer, pp. 80-91, 1999.
[8]Dzeroski S. & Lavrac N. (Eds). *Relational Data Mining*. Springer-Verlag, 2001.
[9]Karalic A.: Linear regression in regression tree leaves. In *Proc. of ISSEK '92 (International School for Synthesis of Expert Knowledge)*, Bled, Slovenia, 1992.
[10]Karalic A.: First Order regression. Ph.D thesis, University of Ljubljana, Slovenia, 1995.
[11]Klosgen W. & May M.:Spatial Subgroup Mining Integrated in an Object-RelationalSpatial Database. In *Principles of Data Mining and Knowledge Discovery,6th European Conference, PKDD 2002*,Elomaa T.,Mannila H. & Toivonen H.(Eds.),Helsinki, Finland, Springer-Verlag , 2002.
[12]Knobbe J., Siebes A.&Van der Wallen D.M.G: Multi-relational decision tree induction.In *Proc. 3rd European Conf. on Principles and Practice of Knowledge Discovery in Databases*,PKDD'99, 1999.
[13] Kramer S.: Structural regression trees. In *Proc. 13th National Conf. on Artificial Intelligence*, 1996.
[14]Leiva H.A.:MRDTL: *A multi-relational decision tree learning algorithm*. Master thesis, University of Iowa, USA, 2002.
[15]Lubinsky D.: Tree Structured Interpretable Regression. In *Learning from Data*, Fisher D. & Lenz H.J. (Eds.), Lecture Notes in Statistics, 112, Springer, 1994.
[16]Malerba D., Appice A., Ceci M. & Monopoli M.: Trading-off versus global effects or regression nodes in model trees. In *Foundations of Intelligent Systems, 13th International Symposium, ISMIS'2002*, Hacid H.S., Ras Z.W. , Zighed D.A. & Kodratoff Y. (Eds.), Lecture Notes in Artificial Intelligence, 2366, Springer, Germany, 2002.
[17]Malerba D., Appice A. & Vacca N.:SDMOQL: An OQL-based Data Mining Query Language for Map Interpretation Tasks. In *Proc. of the EDBT Workshop on "Database Technologies for Data Mining"*, Prague, Czech Republic, 2002.
[18]Morik K. & Scholz M.: *The MiningMart Approach to Knowledge Discovery in Databases*. In Handbook of Intelligent IT, Ning Zhong and Jiming Liu (Eds.),IOS Press,2003, to appear.
[19]Muggleton S., Srinivasan A., King R. & Sternberg M.: Biochemical knowledge discovery using Inductive Logic Programming. In *Proceedings of the first Conference on Discovery Science*, Motoda H. (ed), Springer-Verlag, Berlin, 1998.
[20]Ordonez C. & Cereghini P.: SQLEM: Fast Clustering in SQL using the EM Algorithm. In *Proc. ACM SIGMOD 2000,* Chen W., Naughton J.& Bernstein P.(Eds.), Dallas, USA, vol. 29, 2000.
[21] Orkin. M. & Drogin. R.: *Vital Statistics*. McGraw Hill. New York . 1990.
[22]Quinlan J. R.: Learning with continuous classes, in *Proceedings AI'92*, Adams & Sterling (Eds.), World Scientific, 1992.
[23]Quinlan J. R.: A case study in Machine Leaning, in *Proceedings ACSC-16*, Sixteenth Australian Computer Science Conferences, 1993.
[24] Sarawagi S., Thomas S. & Agrawal R.: Integrating Mining with Relational Database Systems: Alternatives andImplications. In *Proc. ACM SIGMOD'98,* L. Haas and A. Tiwary (Eds), Seattle, *USA*., 1998.
[25]Sattler K. & Dunemann O.: SQL Database Primitives for Decision Tree Classifiers. In *Proc. of the 10th ACM CIKM Int.Conf. on Information and Knowledge Management*, Atlanta, USA, 2001.
[26]Torgo L.: Functional Models for Regression Tree Leaves. In *Proceedings of the 14th International Conference (ICML 97)*, D. Fisher (Ed.), Nashville, Tennessee, 1997.
[27]Wang Y. & Witten I.H.: Inducing Model Trees for Continuous Classes. In *Poster Papers of the 9th European Conf. on Machine Learning (ECML 97)*, M. van Someren, & G. Widmer (Eds.), Prague, Czech Republic,1997.
[28]Wrobel, S.: Inductive logic programming for knowledge discovery in databases. In Dzeroski S. & Lavrac N. (Eds). *Relational Data Mining*. Springer-Verlag, 2001.

Inductive Databases of Polynomial Equations

Sašo Džeroski, Ljupčo Todorovski, and Peter Ljubič

Jožef Stefan Institute, Jamova 39, 1000 Ljubljana, Slovenia

Abstract. Inductive databases (IDBs) contain both data and patterns. Knowledge discovery processes can be viewed as sequences of inductive queries to such databases, where inductive queries can perform both data selection and data mining. Inductive queries consist of conjunctions of primitive constraints that have to be satisfied by target patterns. The constraints can be different for different pattern domains.

Many pattern domains, mostly concerning the discovery of frequent patterns, have been considered separately: these include the domains of frequent itemsets, episodes, Datalog queries, sequences, and molecular fragments. Here we consider inductive databases in the pattern domain of polynomial equations and present a heuristic solver for this domain. We first evaluate the use of this solver on standard regression problems, then proceed to illustrate the use of constraints in discovering dynamics. We finally consider IDBs containing both the pattern domains of equations and molecular fragments, as well as combining them in order to derive QSAR (Quantitative Structure-Activity Relationships) models.

1 Inductive Databases

Inductive databases [7] embody a database perspective on knowledge discovery, where knowledge discovery processes are considered as query processes. In addition to normal data, inductive databases contain patterns (either materialized or defined as views). Data mining operations looking for patterns are viewed as inductive queries posed to the inductive database. In addition to patterns (which are of local nature), models (which are of global nature) can also be considered.

Given an inductive database that contains data and patterns, several different types of queries can be posed. Data retrieval queries use only the data and their results are also data: no pattern is involved in the query. Cross over queries cross over patterns and data in order to obtain new data. In processing patterns, the patterns are queried without access to the data: this is what is usually done in the post-processing stages of data mining. Data mining queries use the data and their results are patterns: new patterns are generated from the data and this corresponds to the traditional data mining step. When we talk about inductive queries, we most often mean data mining queries.

A general formulation of data mining [11] involves the specification of a language of patterns and a set of constraints that a pattern has to satisfy with respect to a given database. The constraints that a pattern has to satisfy can be divided in two parts: language constraints and evaluation constraints. The first

only concern the pattern itself, the second concern the validity of the pattern with respect to a database. Constraints thus play a central role in data mining and constraint-based data mining is now a recognized research topic [1]. The use of constraints enables more efficient induction as well as focusing the search for patterns on patterns likely to be of interest to the end user.

In the context of inductive databases, inductive queries consist of constraints and the primitives of an inductive query language include language constraints (e.g., find association rules with item A in the head) and evaluation primitives. Evaluation primitives are functions that express the validity of a pattern on a given dataset. We can use these to form evaluation constraints (e.g., find all item sets with support above a threshold) or optimization constraints (e.g., find the 10 association rules with highest confidence).

Different types of patterns have been considered in data mining, including frequent itemsets, episodes, Datalog queries, and graphs. Designing inductive databases for these types of patterns involves the design of inductive query languages and solvers for the queries in these languages. For each type of pattern, or pattern domain, a specific solver is designed, following the philosophy of constraint logic programming [2].

While many different types of patterns have been considered in data mining, constraints have been mostly considered in mining frequent patterns, typically frequent itemsets and association rules. Some related tasks, such as mining frequent episodes, Datalog queries, molecular fragments, etc., have also been considered. Here we consider IDBs that contain models in the form of polynomial equations: constrains on these are considered and a heuristic solver is proposed. We first evaluate the use of this solver on standard regression problems, then proceed to illustrate the use of constraints in discovering dynamics. We finally consider IDBs containing both the pattern domains of equations and molecular fragments, as well as combining them in order to derive QSAR (Quantitative Structure-Activity Relationships) models.

2 The pattern domain of polynomial equations

Here we consider the pattern domain of polynomial equations. We first define the language of polynomial equations, then consider syntactic/subsumption constraints on these. We next define several evaluation primitives for equations and finally discuss inductive queries in this domain.

2.1 The language of polynomial equations

In this paper, we will concentrate on polynomial equations that can be used to predict the value of a dependent variable v_d. Given a set of variables V, and a dependent variable $v_d \in V$, a polynomial equations has the form $v_d = P$, where P is a polynomial over $V \setminus \{v_d\}$. A polynomial P has the form $\sum_{i=1}^{r} \text{const}_i \cdot T_i$,

where T_i are multiplicative terms, and const_i are real-valued constants. Each term is a finite product of variables from $V \setminus \{v_d\}$, i.e., $T_i = \prod_{v \in V \setminus \{v_d\}} v^{d_{v,i}}$, where $d_{v,i}$ is (a non-negative integer) degree of the variable in the term. The degree of 0 denotes that the variable does not appear in the term. The sum of

degrees of all variables in a term is called the degree of the term, i.e., $\deg(T_i) = \sum_{v \in V \setminus \{v_d\}} d_{v,i}$.

The degree of a polynomial is the maximum degree of a term in that polynomial, i.e., $\deg(P) = \max_{i=1}^{r} \deg(T_i)$. The length of a polynomial is the sum of the degrees of all terms in that polynomial, i.e., $\text{len}(P) = \sum_{i=1}^{r} \deg(T_i)$.

For example, consider a set of variables $V = \{x, y, z\}$, where z is chosen to be a dependent variable. The term x (that is equivalent to x^1y^0) has degree 1, the term x^2y has degree 3, while x^2y^3 is a term of degree 5. An example polynomial of degree 4 is $1.2x^2y + 3.5xy^3$. An example polynomial equation is $z = 1.2x^2y + 3.5xy^3$. This equation has $r = 2$, $d = 4$ and $len(P) = 7$.

2.2 Syntactic Constraints

We will consider two types of syntactic constraints on equations: parametric constraints and subsumption constraints.

Parametric constraints on polynomial equations include setting upper limits for the degree of a term (in both the LHS and RHS of the equation), as well as the number of terms in the RHS polynomial. For example, one might be interested in equations of degree at most 3 with at most 4 terms. Such parametric constraints can already be taken into account by the equation discovery system LAGRANGE [5].

Of more interest are subsumption constraints, which bear some resemblance to subsumption/generality constraints on terms/clauses in first-order logic. A term T_1 is a sub-term of term T_2 if the corresponding multi-set M_1 is subset of the corresponding multi-set M_2. For example, XY^2 is sub-term of X^2Y^4Z since $\{X, Y, Y\} \subset \{X, X, Y, Y, Y, Y, Z\}$.

The sub-polynomial constraint is defined in terms of the sub-term constraint. Polynomial p_1 is a sub-polynomial of polynomial p_2 if each term in p_1 is a sub-term of some term in p_2. There are two options here: one may, or may not, require that each term in p_1 is a sub-term of a different term in p_2.

In looking for interesting equations, one might consider constraints of the form: LHS is a sub-term of t, t is a sub-term of LHS, RHS is a sub-polynomial of p, and p is a sub-polynomial of RHS. Here t and p are a term and a polynomial, respectively. These set upper and lower bounds on the lattice of equation structures, induced by the relations sub-term and sub-polynomial.

Consider the following constraint: LHS is a sub-term of X^2Y and both XY and Z are sub-polynomials of RHS. The equation $XY = 2X^2Y^2 + 4Z$ satisfies this constraint, under both interpretations of the sub-polynomial constraint. The equation $X^2Y = 5XYZ$, however, only satisfies the constraint under the first interpretation (different terms in p_1 can be sub-terms of the same term in p_2).

2.3 Evaluation Primitives

The evaluation primitives for equations calculate different measures of the degree of fit of the equation to a given dataset/table. Assume that i is an index that runs through records/rows of a database table. Denote with y_i the value of the

LHS of a given equation on record i (actual value of the dependent variable v_d); with $\hat{y_i}$ the value of the RHS as calculated for the same record (predicted value of v_d); and with $\overline{y}$ the mean value of y_i over the table records.

Below we define five measures for the degree of fit for an equation to a dataset. These are the multiple correlation coefficient R (defined as $R^2 = 1 - \frac{\sum_{i=1}^{N}(y_i-\hat{y}_i)^2}{\sum_{i=1}^{N}(y_i-\overline{y})^2}$), the normalized standard deviation S ($S^2 = \frac{\frac{1}{N}\sum_{i=1}^{N}(y_i-\hat{y}_i)^2}{\overline{y}^2+e^{-\overline{y}^2}}$), mean absolute error ($MeanAE = \frac{1}{N}\sum_{i=1}^{N}|\hat{y}_i - y_i|$), maximum absolute error ($MaxAE = \max_{i=1}^{N}|\hat{y}_i - y_i|$), mean square error ($MSE = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2$), and root mean square error ($RMSE = \sqrt{MSE}$). Most of these are well known from statistics. In the machine learning literature, the measure RE, defined as $RE^2 = 1-R^2$ is often used to evaluate the performance of regression approaches.

We will also use a MDL (minimal description length) based heuristic function for evaluating the quality of equations that combines the degree of fit with the complexity of the equation: $\mathrm{MDL}(v_d = P) = len(P)\log N + N\log \mathrm{MSE}(v_d = P)$, where $len(P)$ is the length of a polynomial (the sum of the degrees of all terms in that polynomial) and N number of training examples (data points in D). The second term in the MDL heuristic function introduces a penalty for the complexity of the equation. Thus, the MDL heuristic function introduces a preference toward simpler equations.

2.4 Inductive queries in the domain of equations

Inductive queries are typically conjunctions of constraints. The primitive constraints typically are evaluation and language constraints. Evaluation constraints set thresholds on acceptable values of the evaluation primitives: $M(e, D) < t$; $M(e, D) > t$, where t is a positive constant/threshold and M is one of the measures defined above.

Instead of evaluation constraints one can consider optimization constraints. Here the task is to find (the n best) e so that $M(e, D)$ is maximal / minimal. Language constraints, as discussed above, can be parametric and/or subsumption constraints.

It is rarely the case that an inductive query consists of a single constraint. Most often, at least one syntactic and at least one evaluation or optimization constraint would be a part of the query. For example, we might look for the equations, where the LHS is sub-polynomial of X^2Y^3 and $X + Z$ is a sub-polynomial of the RHS, which have the highest multiple correlation coefficient.

3 A heuristic solver for the domain polynomial equations

While most systems for discovering frequent patterns look for all frequent patterns that satisfy a given set of constraints, most approaches to predictive modeling perform heuristic search through the space of possible models. Here we present CIPER (which stands for Constrained Induction of Polynomial Equations for Regression) that heuristically searches through the space of possible

equations for solutions that satisfy the given constraints. CIPER employs beam search through the space of polynomial equations.

Table 1. A top-level outline of the CIPER beam search algorithm.

```
   procedure CIPER(D, v_d, C, b)
 1 E_0 = simplest polynomial equation (v_d = const )
 2 E_0.MDL = FITPARAMETERS(E_0, D)
 3 Q = {E_0}
 4 repeat
 5    Q_r = {refinements of equation structures in Q
                 that satisfy the language constraints C}
 6    foreach equation structure E ∈ Q_r do
 7       E.MDL = FITPARAMETERS(E, D)
 8    endfor
 9    Q = {best b equations from Q ∪ Q_r according to MDL }
10 until Q unchanged during the last iteration
11 print Q
```

The algorithm, shown in Figure 1, takes as input the training data D, i.e, the training examples, each of them containing the measurements of the observed (numerical) system variables, and a designated dependent variable v_d. In addition, a set of language constraints C can be specified. The output of CIPER consists of the b polynomial equations for v_d that satisfy the constraints C and are best wrt the data D according to the MDL heuristic function defined in the previous section.

Before the search procedure starts, the beam Q is initialized with the simplest possible polynomial equation of the form $v_d = \text{const}$. The value of the constant parameter const is fitted against the training data D using linear regression and the MDL heuristic function is calculated. In each search iteration, the refinements of the equations in the current beam are computed first and checked for consistency with the specified language constraints C. The refinement operator increases the length of a given equation by one, either by adding a new linear term or by adding a variable to an existing term in the equation. Finally, at the end of each search iteration, only the best b equations, according to the MDL heuristic function, are kept in the beam. The search proceeds until the beam remains unchanged during the last iteration.

The refinement operator used is not optimal: the same polynomial equation can be generated through different paths. While an optimal refinement operator is desired for complete/exhaustive search, it may prevent the generation of good equations in greedy heuristic search. We thus choose to use the above in-optimal operator and check for duplicates when adding new equations to the beam.

Each polynomial equation structure considered during the search contains a number of generic constant parameters (denoted by const_i). In order to evaluate the quality of an equation, the values of these generic constants have to be fitted against training data consisting of the observed values of the variables in

V. Since the polynomial equations are linear in the constant parameters, the standard linear regression can be used.

The quality of the obtained equation is usually evaluated using an evaluation primitive, i.e., a degree of fit measure that measures the discrepancy between the observed values of v_d and the values predicted using the equation. One such measure is mean squared error (MSE), defined in the previous section. Other measure that beside degree of fit takes into account the complexity of the induced equation and introduce preference toward simpler equations is MDL.

CIPER can also discover differential equations. Just like LAGRANGE, it can introduce the time derivatives of system variables by numerical derivation. It can then look for equations of the form $\dot{x_i} = P(x_1, x_2, ..., x_n)$, where $\dot{x_i}$ denotes the time derivative of x_i and $P(x_1, x_2, ..., x_n)$ denotes a polynomial of $x_1, x_2, ..., x_n$.

4 Polynomial equations on standard regression datasets

Equation discovery approaches, such as LAGRANGE [5], have been typically evaluated in terms of successful rediscovery of quantitative laws. In particular, data generated from known laws/models has been used. The emphasis has mainly been on the comprehensibility and general validity of the laws found, rather than their predictive power. One of the reasons for this has been the prohibitive computational complexity of applying exhaustive search to general regression problems involving many variables and potentially complex laws. Having a heuristic approach for equation discovery, we evaluate it on a number of regression datasets.

Table 2. The 14 regression UCI datasets and their characteristics (left four columns). The performance of CIPER in terms of RE, as compared to three other regression approaches: linear regression (LR), model trees (MT), and regression trees (RT) (right four columns).

Dataset	Exs	Vars	NERPV	CIPER	LR	MT	RT
autoprice	159	16	3.24	0.15	0.23	0.15	0.32
baskball	96	5	4.87	0.61	0.67	0.63	0.78
bodyfat	252	15	2.22	0.03	0.03	0.03	0.11
elusage	55	3	7.56	0.18	0.23	0.28	0.44
fruitfly	125	5	3.94	1.01	1.10	1.03	1.01
housing	506	14	1.24	0.19	0.29	0.17	0.28
mbagrade	61	3	6.97	0.78	0.83	0.83	1.05
pollution	60	16	7.06	0.55	0.55	0.42	0.77
pwLinear	200	11	2.68	0.15	0.25	0.11	0.33
quake	2178	4	0.35	1.00	1.00	0.99	1.00
sensory	576	12	1.11	0.89	0.87	0.75	0.85
strike	625	7	1.04	0.95	0.84	0.83	0.87
veteran	137	8	3.66	0.90	0.92	0.88	0.91
vineyard	52	4	7.89	0.29	0.43	0.48	0.73
Average RE				0.55	0.59	0.54	0.67

We take 14 regression datasets from the UCI repository, listed in the left part Table 2. In addition to the number of examples (data points) and variables, we also list the reduction of MSE (in percent) necessary to counterbalance an increase in $len(P)$ of 1 in the MDL heuristic (Needed Error Reduction Per Variable - NERPV).

We perform ten-fold cross-validation to evaluate the performance of our approach. The performance is measured in terms of RE. We compare the performance of our approach to the performance of linear regression (with no feature selection, no feature elimination), model trees, and regression trees (all of these within WEKA). The results are given in the right part of Table 2.

Note that a beam of 16 was used in CIPER. Our approach performs better than linear regression and regression trees and comparably to model trees. We should note that this is achieved using a single equation/model over the entire instance space, rather than a piece-wise model (as in model trees).

It is interesting to compare the complexity of the equations found by our approach to that of model trees. The complexity of model trees can be measured as the number of parameters/constants in them, including the thresholds in internal nodes and the coefficients in linear regressions in the leaves. The complexity of equations can be measured in several ways ($d + r$, $len(P)$), including the number of coefficients (where we count the degree of a variable in a term as a coefficient). The average total number of parameters in model trees is 16.83 and in regression trees 13.72. The average d, r and $len(P)$ are 1.64, 4.93 and 6.00, respectively. The polynomial equations discovered thus compare favorably in terms of complexity to regression trees and model trees and are about the same level with linear regression (which has on the average 11.93 parameters).

5 Using constraints in modeling chemical reactions

To illustrate the use of constraints in discovering dynamics, we address the task of reconstructing a partially specified network of chemical reactions. The network is shown in Fig. 1: the part of the network given in bold is assumed to be known, while the part shown in gray is unknown (except for the fact that x_6 and x_7 are involved in the network). This is a task of revising an equation-based model.

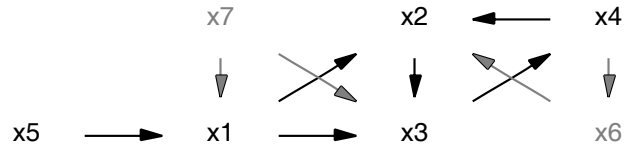


Fig. 1. A partially specified network of chemical reactions.

A network of chemical reactions can be modeled with a set of polynomial differential equations (see, e.g., [8]). The transformation of a network to a set of differential equations is performed in the following manner. The reaction rate is proportional to the concentrations of inputs involved in the reaction. For

example, consider the reaction $\{x_5, x_7\} \rightarrow x_1$, on the left-hand side of Figure 1. It takes x_5 and x_7 as inputs, therefore the corresponding term in the equations is $x_5 \cdot x_7$. The reaction rate influences the rate of change of all (input and output) compounds involved in the equations. Therefore, the term $x_5 \cdot x_7$, will appear in the equations for x_1, x_5, and x_7. In the equation for the output compound x_1 the term positively influences the change rate, while in the equations for the input compounds the term negatively influences the change rate.

Following the algorithm outlined above, the following set of differential equations can be composed and used for modeling the network from Figure 1:

$$\begin{aligned}
\dot{x_1} &= 0.8 \cdot x_5 \cdot x_7 - 0.5 \cdot x_1 - 0.7 \cdot x_1 \cdot x_2 \cdot x_7 \\
\dot{x_2} &= 0.7 \cdot x_1 + 0.2 \cdot x_4 + 0.1 \cdot x_4 \cdot x_6 - 0.3 \cdot x_1 \cdot x_2 \cdot x_7 \\
\dot{x_3} &= 0.4 \cdot x_1 + 0.3 \cdot x_1 \cdot x_2 \cdot x_7 - 0.2 \cdot x_3 \\
\dot{x_4} &= 0.5 \cdot x_3 - 0.7 \cdot x_4 \cdot x_6 \\
\dot{x_5} &= -0.6 \cdot x_5 \cdot x_7 \\
\dot{x_6} &= 0.2 \cdot x_4 - 0.8 \cdot x_4 \cdot x_6 \\
\dot{x_7} &= -0.1 \cdot x_1 \cdot x_2 \cdot x_7 - 0.1 \cdot x_5 \cdot x_7
\end{aligned}$$

These equations were simulated for 1000 time steps of 0.01 from a randomly generated initial state (where the value of each variable was randomly chosen from the interval (0,1)). This provides a trace of the behavior of the 7 system variables over time, suitable for discovering dynamics.

The domain of modeling networks of chemical reactions lends itself naturally to the use of constraints in polynomial equation discovery. On one hand, parametric constraints have a natural interpretation. A limit on r, the number of terms in an equation, corresponds to a limit on the number of reactions a compound is involved in. A limit on d, the degree of terms, corresponds to a limit on the number of compounds that are input to a chemical reaction. On the other hand, subsumption constraints can also be used in a natural way. A partially specified reaction network gives rise to equations that involve subpolynomials of the polynomials modeling the entire network.

If only the bold part of the network was present, the following equations could be used to model its behavior.

$$\begin{aligned}
\dot{x_1} &= -const \cdot x_1 + const \cdot x_5 - const \cdot x_1 \cdot x_2 \\
\dot{x_2} &= const \cdot x_1 + const \cdot x_4 - const \cdot x_1 \cdot x_2 \\
\dot{x_3} &= const \cdot x_1 + const \cdot x_1 \cdot x_2 - const \cdot x_3 \\
\dot{x_4} &= const \cdot x_3 - const \cdot x_4 \\
\dot{x_5} &= -const \cdot x_5
\end{aligned}$$

The knowledge of the partial network can be used to constrain the search through the space of possible equations. The polynomials in the equations for $\dot{x_1}$ ... $\dot{x_5}$ in the partial network should be subpolynomials for the corresponding equations in the complete network. These subpolynomial constraints were given

to CIPER together with the behavior trace for all 7 variables. No subsumption constraints were used for the equations defining $\dot{x_6}$ and $\dot{x_5}$. No parametric constraints were used for any of the equations. Beams of size 64 and 128 were used in the experiments.

CIPER then successfully reconstructs the equations for the entire network, i.e., for each of the 7 system variables, for each of the two beam sizes. Discovery without constraints, however, fails for two of the equations. If we provide CIPER only with the behavior trace and no constraints, it fails to reconstruct the equations for $\dot{x_1}$ (beam 128) and $\dot{x_2}$ (for both beams) correctly. In addition, the unconstrained search inspects more equations than the constrained (for $\dot{x_2}$ and beam 128, 18643 and 12901 equations were considered). For comparison, exhaustive search through all equations with $d \leq 3$ and $r \leq 4$ would have to consider 637393 equations.

6 Towards inductive databases for QSAR

Here we first describe the pattern domain of molecular fragments. We then proceed with a proposal of how to integrate the pattern domains of equations and molecular fragments in order to obtain an inductive database for QSAR (Quantitative Structure-Activity Relationships). Preliminary experiments in the domain of predicting biodegradability, illustrating how the two domains can be combined, are presented.

6.1 The pattern domain of molecular fragments

Here we briefly summarize the pattern domain of molecular fragments, introduced by Kramer and De Raedt [9], in terms of the syntactic constraints and evaluation primitives considered. The system MolFea, which implements a solver for this pattern domain, looks for interesting molecular fragments (features) in sets of molecules. Interestingness is defined as a conjunction of primitive constraint that the fragment has to satisfy.

A molecular fragment is defined as a sequence of linearly connected atoms. For instance, $o{-}c = o$ is a fragment meaning: "an oxygen atom with a single bond to a carbon atom with a double bond to an oxygen atom". In such expressions 'c', 'n', 'o', etc. denote elements, and '$-$' denotes a single bond, '$=$' a double bond, '#' a triple bond, and '$\sim$' an aromatic bond. Only non-hydrogen atoms are considered. Fragments are partially ordered by the relation "is more general than"/"is a subsequence of": when fragment g is more general than fragment s, one writes $g \leq s$.

The primitive language constraints that can be imposed on unknown target fragments f are of the form $f \leq p$, $p \leq f$, $\neg(f \leq p)$ and $\neg(p \leq f)$, where f is the unknown target fragment and p is a specific pattern. This type of primitive constraint denotes that f should (not) be more specific (general) than the specified fragment p. E.g., the constraint '$c = o$' $\leq f$ specifies that f should be more specific than '$c = o$', i.e., that f should contain '$c = o$' as a subsequence.

The evaluation primitive $freq(f, D)$ denotes the relative frequency of a fragment f on a set of molecules D. The relative frequency is defined as the percentage of molecules (from D) covered (by f). Evaluation constraints can be defined by specifying thresholds on the frequency or relative frequency of a fragment on a dataset: $freq(f, D_1) \leq t$ and $freq(f, D_1) \geq t$ denote that the relative frequency of f on D should be larger than (resp. smaller than) or equal to t. For example, the constraint $freq(f, Pos) \geq 0.95$ denotes that the target fragments f should have a minimum relative frequency of 95% on the set of molecules Pos.

The primitive constraints defined above can conjunctively be combined in order to declaratively specify the target fragments of interest. Note that the conjunction may specify constraints w.r.t. any number of datasets, e.g. imposing a minimum frequency on a set of active molecules, and a maximum one on a set of inactive ones. For example, the constraint ($'c = o' \leq f) \wedge \neg(f \leq 'c-c-o-c = o')$ $\wedge freq(f, Deg) \geq 0.95 \wedge freq(f, NonDeg) \leq 0.05)$ queries for all fragments that include the sequence $'c = o'$, are not a subsequence of $'c - c - o - c = o'$, have a frequency on Deg that is larger than 95 percent and a frequency on $NonDeg$ that is smaller than 5 percent.

6.2 Combining molecular fragments and equations in an inductive database for QSAR

The basic idea of our proposal is to consider the pattern domains of equations and molecular fragments in a single inductive database. One could then easily use the results of one inductive query (e.g., the set of interesting features resulting from a MolFea query) as input to another inductive query (e.g. to find a QSAR equation for biodegradability). This is of interest as QSAR models most often take the form of equations and molecular fragments are often used as features.

This is similar to what Kramer and De Raedt [9] do. They look for interesting features with MolFea, then use these for predictive modeling (classification with a number of data mining approaches). However, no constraints are used in their predictive modeling phase. Also, the data mining approach that performs best in their setup is that of support vector machines, which does not yield interpretable models.

Each of the two pattern domains offers potentially useful functionality. Taken by themselves, equations are the approach of choice for QSAR modeling. While non-linear transformations of bulk features are sometimes used, most often linear equations are sought. Our constraint-based approach to equation discovery would allow the QSAR modeler to pose inductive queries that focus the search on interesting equations, such as: "find me the best equation that involves featureX", supposing featureX is of interest to the QSAR modeler.

Also, molecular fragments (or in general substructures) are often used as features in QSAR modeling. Using feature mining in the pattern domain of molecular fragments, the QSAR modeler can find patterns involving substructures of special interest. These can then be used in QSAR models.

The basic form of exploiting the connection between the two pattern domains is to use the molecular fragments found by MolFea as input for a data

mining query looking for QSAR equations. Here one can exploit the subpolynomial constraints in the equations pattern domain to ask for equations that contain a specific molecular fragment: the fragment in question should be a subpolynomial of the RHS of the equation sought. However, additional constraints can be defined. For example, one may look for equations that involve a subfragment or superfragment of a given fragment, rather than just the fragment itself. We have implemented the latter in our system CIPER as an extension of the subpolynomial/superpolynomial constraint.

At present, we assume all frequent fragments are given and look for equations that involve the given equations and satisfy the constraints. As an illustration, in the biodegradability dataset, we may be interested in the two best equations that contain only one feature that is a superfragment of '$c = o$'. The equations $logHLT = 6.9693 - 1.19013 * c = o$ and $logHLT = 6.91524 - 1.24286 * c - c = o$ are returned as a result of this query.

When we are looking for equations involving binary variables, such as the presence/absence of a molecular fragment in a molecule, we should take into account that higher degrees of such variables are identical to the variables themselves. Thus the higher degrees of molecular fragment should not appear in equations. This can drastically reduce the space of possible equations.

At present, the above is not taken into account explicitly. However, the search heuristic used in CIPER, which takes into account equation complexity (punishes complex equations) prevents equations with higher degrees of fragment features from being chosen. This because such equations have the same accuracy/degree of fit as the equation with a linear degree of the fragment and higher complexity.

Other constraints might arise from the properties of molecular fragments and generality/specificity relationships among them, which might be used to reduce/heuristically prune the space of possible QSAR equations. For example, when refining a fragment, i.e., making it more specific, the value of the corresponding feature can only go from one to zero for any molecule considered. This can be used to bind the possible error reduction and also the maximum heuristic value an equation can achieve.

Originally, our constraint-based approach to equation discovery was meant for regression problems (with continuous class variable). However, we can easily adapt it to classification problems where the features are numeric (including binary features). We can use CIPER to perform classification via regression (multi-response polynomial regression), an approach shown to perform well in many cases.

6.3 Experiments on the biodegradability dataset

The QSAR application that we consider here is the one of predicting biodegradability of compounds [4]. The database used was derived from the data in the handbook of degradation rates [6] and contains the chemical structure and measured (or estimated) degradation rates for 328 chemicals. In our study we focus on aqueous biodegradation half life time (HLT) in aerobic conditions. The target variable (denoted logHLT) is the natural logarithm of the arithmetic mean of

the low and high estimate of the HLT for aqueous biodegradation in aerobic conditions, measured in hours. Two discretized versions (2-class and 4-class) of logHLT were also considered.

A global feature of each chemical is its molecular weight. This was included in the data. Another global feature is logP, the logarithm of the compound's octanol/water partition coefficient, used also in the mutagenicity application. This feature is a measure of hydrophobicity, and can be expected to be important since we are considering biodegradation in water. The two global features were used in addition to the discovered molecular fragments in the QSAR models developed.

Kramer and De Raedt [9] used the two-class version of the biodegradability dataset to mine for interesting features. They looked for features that are above a prespecified minimum frequency threshold and below a maximum threshold. No generality constraints were used. The resulting features (fragments) together with MolWeight and log P were then used for predictive modeling (classification with a number of data mining approaches, of which support vector machines performed best).

For our experiments, the features generated with thresholds of minimum 0.05 and maximum 0.95 were kindly provided by Stefan Kramer. A total of 124 fragments were found by MolFea. Of these, only 69 are distinct, in the sense that there is at least one compound where one fragment is present and the other is not. The major reason for getting duplicates is that MolFea reported all fragments that satisfied the frequency thresholds and not the minimal (maximal) ones: duplicates can be avoided by taking the borders of the versionspace produced by MolFea. Only 69 distinct fragments were taken as input for equation discovery.

Table 3. The equation generated by CIPER on the biodegradability dataset.

```
logHLT = 0.0645738*logP - 0.825858*c=o + 0.374477*logP*cl + 0.487245*c-n
         + 2.43385*c~c~c~c~c~c~c~c~c~c~c~c~c~c - 0.529246*c~c~c~c~c
         + 0.757922*n=o + 0.570323*c-c-c~c~c~c~c~c
         - 0.632581*c-c-c-o + 0.817581*c-o-c - 0.621152*c-o
         + 0.00231708*MolWeight + 5.94176
```

Taking the 69 fragments, logP and MolWeight as the independent and logHLT (or the discretization thereof) as the dependent variable, we applied our system for equation discovery CIPER to the biodeg dataset. For the continuous version, the equation in Table 3 was generated by CIPER (with beam 4). For the two-class and four-class version, classification via regression was performed: as many equations were produced as the number of classes.

The predictive performance of CIPER, estimated by 10-fold cross validation, was also measured (Table 4). It was compared to the performance of three other algorithms: J48, Linear Regression and M5. For the latter two, comparison was done for both the continuous (regression) and the discrete (classification via regression) versions of the dataset.

Table 4. The predictive performance of CIPER on the biodegradability dataset, estimated by 10-fold cross validation, as compared to several other learning algorithms.

Data version / Algorithm	J48 (Class via)	LinReg (Class via)	M5 (Class via)	CIPER
2 class (acc.)	71.34	74.09	78.05	78.66
4 class (acc.)	56.40	52.24	54.57	56.40
continuous (RE)		0.627	0.560	0.576

CIPER performs much better than J48 for the two-class version and the same for the four-class version. It performs better than linear regression (higher accuracy, lower error / RE). It performs better than M5 for classification via regression and slightly worse for regression. Overall, CIPER performs best. In addition, the equation produced can be easily interpreted (e.g., three fused aromatic rings greatly contribute to longer degradation time).

7 Summary

Here we have considered the problem of predictive modeling via polynomial equations within the framework of inductive databases (IDBs). We have defined primitives for the pattern domain of polynomial equations, including language constraints (such as sub-polynomial), evaluation primitives, evaluation and optimization constraints. We have also shown how to combine such primitives to form inductive queries in this pattern domain.

We have then presented a heuristic solver for data mining queries in this pattern domain. The algorithm CIPER performs beam search through the space of polynomial equation that satisfy a set of given language constraints and reports the beam of equations that satisfy the evaluation constraints or best satisfy the optimization constraints. It uses a refinement operator derived from the subsumption (sub-polynomial) relationship between polynomials and a heuristic function that takes into account both the accuracy and complexity of equations.

We have finally illustrated the use of the developed approach in three different areas. First we have applied CIPER to standard regression datasets, where it performs competitively to existing regression methods while producing smaller models. We have then shown that constraint-based discovery of polynomial equations is suitable for modeling the dynamics of chemical reaction networks, since the language constraints in our pattern domain naturally correspond to complexity limits on reaction networks. Finally, we have indicated how the pattern domains of equations and molecular fragments can be combined into an IDB for QSAR and illustrated the used of such an IDB on the problem of predicting biodegradability.

8 Discussion

The present paper emphasizes the use of equations as predictive models in data mining. Regression equations are commonly used for predictive modeling in statistics, but receive considerably less attention in data mining. While equation discovery [10] is a recognized research topic within machine learning, it has been previously mainly used to rediscover quantitative scientific laws, with an

emphasis on the comprehensibility and general validity of the laws found, rather than their predictive power. Here we show that equation discovery can build models that have similar predictive power and lower complexity as compared to state of the art regression approaches.

In the framework of inductive databases (IDBs), different types of patterns have been considered within different so-called pattern domains [2]. Most considered pattern domains concern the discovery of frequent patterns, such as frequent itemsets, episodes, Datalog queries and sequences. One of the main contributions of this paper is that global predictive models (equations) have been considered in the framework of IDBs, following the same general structure of a pattern domain as for the discovery of frequent patterns. Predictive modeling lends itself to being described in terms of the same types of primitives: language constraints, evaluation primitives, evaluation and optimization constraints, as shown by defining such primitives for the pattern domain of equations.

Considering predictive models in the same framework as frequent patterns brings out the contrast between the practice of mining patterns and models. Techniques for mining frequent patterns typically aim at finding all frequent patterns that satisfy a user provided set of constraints (such as minimal frequency). On the other hand, predictive modeling techniques heuristically search for a single model trying to maximize predictive accuracy. (usually not taking into account any other constraints).

In the frequent pattern setting, frequency constraints enable pruning of the space of candidate patterns and using constraints can mean drastic improvements in terms of efficiency. The monotonicity/anti-monotonicity of such constraints, used in conjunction with a generality relation on the patterns, is crucial in this respect. In mining for models, one typically uses constraints (like accuracy) that are neither monotonic nor anti-monotonic. In such cases, the search space over the possible models can still be structured according to the constraints, but effective pruning is no longer possible. As a consequence, heuristic (rather than complete) solvers have to be designed, as was the case in our pattern domain of polynomial equations.

The use of constraints in predictive data mining is less likely to be driven primarily by efficiency considerations. This is because constraints there are less likely to be monotone/anti-monotone, i.e., to enable efficient pruning. In this context, it is important that the constraints are intuitively understandable for the domain experts and have natural interpretation in the domain of use. This has been illustrated by the use of constraints in modeling the dynamics of networks of chemical reactions.

So far, in the framework of IDBs, different types of patterns have been considered separately, within different pattern domains. Here we have considered an IDB with two different pattern domains, equations and molecular fragments: the latter has been considered in an IDB approach targeted at bio- and chemoinformatics applications [3]. Molecular fragments are of interest for QSAR modeling, as the discovered fragments can be used as features in QSAR models. We have presented preliminary experiments showing how the pattern domains

of equations and molecular fragments can be combined by posing queries that involve constraints both on the QSAR equations and the molecular fragments involved therein.

We believe that both global models (like equations) and local patterns (like molecular fragments) will need to be considered in IDBs. Moreover, different types of patterns will need to be used in the same IDB for practical applications, such as QSAR (and other applications in bioinformatics). To our knowledge, global models in the form of equations have so far not been considered in IDBs (even though they are routinely used in practical applications, such as QSAR) and neither have combinations of local patterns and global models.

9 Further work

Concerning work on IDBs with predictive models, we have just began to scratch the surface. Even for the pattern domain of equations there are several directions for further work. One concerns the definition, implementation and use of similarity constraints, which allow the formulation of queries such as: "find the equation most similar to e, which has mean absolute error smaller than x". Another concerns the extension of the pattern domain of equations towards general equations (non-polynomial ones). Here the connection to grammar-based equation discovery [12] might prove useful. Finally, extensions towards other types of regression models (e.g. model trees) as well as classification models would be in order. The latter would be especially relevant for predictive regression (modeling) applications, such as the ones considered in Section 4.

In the domain of modeling reaction networks, an immediate direction for further work concerns the further formulation and exploitation of constraints. For example, instead of a partially specified network with missing reactions, one might consider a given maximal network of reactions and look for a subnetwork. In this case, superpolynomial constraints might be used to focus/constrain the search for equations. Experiments on real data are needed to thoroughly evaluate our approach.

Concerning the work towards IDBs for QSAR, much work remains to be done as well. We have not even completely and formally specified the language of queries involving both fragments and equations that can be posed to (and solved by) our current implementation. As immediate direction for further work, we thus plan a formal specification of the inductive query language for QSAR that we hint at in this paper.

On the technical side, further exploration of the possibilities for integration between fragment and equation discovery deserves attention. At present, tighter integration seems possible and desirable. This could involve, for example, interleaving the processes of feature mining and equation discovery. Both could, in principle, provide useful feedback for each other.

The evaluation on one dataset without significance tests of the differences in performance can be hardly considered sufficient. Evaluation on more datasets is thus needed and planned. But even more than an evaluation on a larger number

of datasets, an evaluation in cooperation with a domain expert in QSAR and bioinformatics is needed in order to determine what scenarios of usage of IDBs are needed, feasible, and/or desirable from an application point of view.

Acknowledgments

Thanks to Stefan Kramer for interesting discussions on the topic of this paper and for providing the MolFea generated features for the biodegradability dataset. We acknowledge the support of the cInQ (Consortium on discovering knowledge with Inductive Queries) project, funded by the European Commission under contract IST-2000-26469.

References

1. R. Bayardo. Constraints in data mining. *SIGKDD Explorations*, 4(1), 2002.
2. De Raedt, L. Data mining as constraint logic programming. In *Computational Logic: From Logic Programming into the Future (In honor of Bob Kowalski)*. Springer, Berlin, 2002.
3. L. De Raedt and S. Kramer. Inductive databases for bio and chemoinformatics. In P. Frasconi, R. Shamir (editors), *Artificial Intelligence and Heuristic Methods for Bioinformatics*. IOS Press, Amsterdam, 2002. To appear.
4. S. Džeroski, H. Blockeel, B. Kompare, S. Kramer, B. Pfahringer, and W. Van Laer. Experiments in predicting biodegradability. In *Proc. Ninth International Conference on Inductive Logic Programming*, pages 80–91. Springer, Berlin, 1999.
5. S. Džeroski and L. Todorovski. Discovering dynamics: from inductive logic programming to machine discovery. *Journal of Intelligent Information Systems*, 4:89–108, 1995.
6. Howard, P.H., Boethling, R.S., Jarvis, W.F., Meylan, W.M., and Michalenko, E.M. 1991. *Handbook of Environmental Degradation Rates*. Lewis Publishers.
7. T. Imielinski and H. Mannila. A database perspective on knowledge discovery. *Communications of the ACM*, 39(11):58–64, 1996.
8. Koza, J.R., Mydlowec, W., Lanza, G., Yu, J., and Keane, M.A. Reverse engineering of metabolic pathways from observed data using genetic programming. In *Proc. Sixth Pacific Symposium on Biocomputing*, pages 434-445. World Scientific, Singapore, 2001.
9. S. Kramer and L. De Raedt. Feature construction with version spaces for biochemical applications. In *Proc. Eighteenth International Conference on Machine Learning*, pages 258–265. Morgan Kaufmann, San Francisco, 2001.
10. Langley, P., Simon, H. A., Bradshaw, G. L., & Żythow, J. M. (1987). *Scientific discovery*. Cambridge, MA: MIT Press.
11. H. Mannila and H. Toivonen. Levelwise search and borders of theories in knowledge discovery. *Data Mining and Knowledge Discovery*, 1(3):241–258, 1997.
12. Todorovski, L., and Džeroski, S. Declarative bias in equation discovery. In *Proc. Fourteenth International Conference on Machine Learning*, pages 376–384. Morgan Kaufmann, San Francisco, CA, 1997.

Towards Optimizing Conjunctive Inductive Queries

Johannes Fischer and Luc De Raedt

Inst. für Informatik
University of Freiburg
Georges Koehler Allee 79
D-79110 Freiburg, Germany
deraedt@informatik.uni-freiburg.de
jfischer@informatik.uni-freiburg.de

Abstract. Conjunctive inductive queries are queries to an inductive database that can be written as a conjunction of a monotonic and an anti-monotonic subquery. We introduce the conjunctive inductive query optimization problem, which is concerned with minimizing the cost of computing the answer set to a conjunctive query. In the optimization problem, it is assumed that there are costs c_a and c_m associated to evaluating a pattern w.r.t. a monotonic and an anti-monotonic subquery. The aim is then to minimize the total cost needed to compute all solutions to the query. Secondly, we present an algorithm that aims at optimizing conjunctive inductive queries in the context of the pattern domain of strings. It employs a data structure based on version space trees, a combination of suffix tries and version spaces. Even though the approach is presented in the pattern domain of strings, the ideas are more widely applicable. Finally, the presented algorithm is empirically evaluated on a challenging data set in computational biology.

1 Introduction

Many data mining problems address the problem of finding a set of patterns that satisfy a constraint. Formally, this can be described as the task of finding the set of patterns $Th(Q, \mathcal{D}, \mathcal{L}) = \{\phi \in \mathcal{L} \mid Q(\phi, \mathcal{D})$, i.e. those patterns ϕ satisfying query Q on database $\mathcal{D}\}$. Here $\mathcal{L}$ is the language in which the patterns or rules are described and Q is a predicate or constraint that determines whether a pattern ϕ is a solution to the data mining task or not [18]. This framework allows us to view the predicate or the constraint Q as an *inductive query* to an *inductive database system*. It is then the task of the inductive database management system to efficiently generate the answers to the query [7].

In this paper, we will study inductive queries that can be written as the conjunction $Q_a \wedge Q_m$ of an anti-monotonic and a monotonic subquery. Conjunctive inductive queries of this type have been studied in various contexts, cf. [5, 6, 17, 4]. One important result is that their solution space $Th(Q, \mathcal{D}, \mathcal{L})$ is convex, which is related to the well-known concept of version spaces [19] and boundary sets [18].

This fact is exploited by several pattern mining algorithms. Indeed, consider a constraint such as $freq(\phi, \mathcal{D}) \geq 10$. If a certain pattern ψ does not satisfy this constraint, then no pattern that is more specific than ψ can satisfy the constraint. This follows from the monotonicity property of the constraint and shows that testing one pattern provides information about whether a corresponding set of patterns may belong to the solution space or not. At this point, the question arises as to whether we can exploit the analogy with concept-learning further in order to obtain strategies that would minimize the cost of evaluating the patterns. The key contribution of this paper is that we introduce an algorithm for computing the set of solutions $Th(Q, \mathcal{D}, \mathcal{L})$ to a conjunctive inductive query that aims at minimizing the cost of evaluating patterns w.r.t. the primitive constraints in the inductive query. More precisely, we assume that the query is of the form $Q_a \wedge Q_m$ where Q_a is anti-monotonic and Q_m is monotonic, that there is a cost c_m and c_a associated to testing whether a pattern satisfies Q_m, resp. Q_a, and we aim at minimizing the total cost of computing $Th(Q, \mathcal{D}, \mathcal{L})$. The algorithm that we introduce builds upon the work by [6] that has introduced an effective data structure, called the version space tree, and algorithms for computing $Th(Q, \mathcal{D}, \mathcal{L})$ for string patterns. In the present paper, we modify this data structure into the partial version space tree and also present an algorithm that employs it in order to minimize the total evaluation cost. The approach is also empirically evaluated on the task of finding patterns in a computational biology data set. It is compared to the more traditional levelwise approach introduced in [6].

The paper is organized as follows. Section 2 introduces the problem of conjunctive query optimization. Section 3 presents a data structure and algorithm for tackling it. Section 4 reports on an experimental evaluation and finally, Sect. 5, concludes and touches upon related work.

2 Conjunctive Inductive Queries

In this section, we define conjunctive inductive queries as well as the pattern domain of strings. Our presentation closely follows that of [6].

A pattern language $\mathcal{L}$ is a formal language for specifying patterns. Each pattern $\phi \in \mathcal{L}$ matches (or covers) a set of examples ϕ_e, which is a subset of the universe $\mathcal{U}$ of possible examples.

Example 1. Let Σ be a finite alphabet and $\mathcal{U}_\Sigma = \Sigma^*$ the universe of all strings over Σ. We will denote the empty string with ϵ. The traditional pattern language in this domain is $\mathcal{L}_\Sigma = \mathcal{U}_\Sigma$. A pattern $\phi \in \mathcal{L}_\Sigma$ covers the set $\phi_e = \{\psi \in \Sigma^* \mid \phi \sqsubseteq \psi\}$, where $\phi \sqsubseteq \psi$ denotes that ϕ is a substring of ψ.

One pattern ϕ is *more general* than a pattern ψ, written $\phi \succeq \psi$, if and only if $\phi_e \supseteq \psi_e$.

A pattern *predicate* defines a primitive property of a pattern, usually relative to some data set $\mathcal{D}$ (a set of examples), and sometimes other parameters. For any given pattern, it evaluates to either *true* or *false*.

We now introduce a number of pattern predicates that will be used for illustrative purposes throughout this paper. Our first pattern predicates are very general in that they can be used for arbitrary pattern languages:

- $\mathsf{minfreq}(\phi,n,\mathcal{D})$ evaluates to true iff ϕ is a pattern that occurs in database $\mathcal{D}$ with frequency at least $n \in \mathbb{N}$. The frequency $f(\phi,\mathcal{D})$ of a pattern ϕ in a database $\mathcal{D}$ is the (absolute) number of examples in $\mathcal{D}$ covered by ϕ. Analogously, the predicate $\mathsf{maxfreq}(\phi,n,\mathcal{D})$ is defined.
- $\mathsf{ismoregeneral}(\phi,\psi)$ is a predicate that evaluates to true iff pattern ϕ is more general than pattern ψ. Dual to the ismoregeneral predicate one defines the ismorespecific predicate.

The following predicate is an example predicate tailored towards the specific domain of string-patterns over $\mathcal{L}_\Sigma$.

- $\mathsf{length\_atmost}(\phi,n)$ evaluates to true for $\phi \in \mathcal{L}_\Sigma$ iff ϕ has length at most n. Analogously the $\mathsf{length\_atleast}(\phi,n)$ predicate is defined.

In all the preceding examples the pattern predicates have the form $\mathsf{pred}(\phi,params)$ or $\mathsf{pred}(\phi,\mathcal{D},params)$, where *params* is a tuple of parameter values, $\mathcal{D}$ is a data set and ϕ is a pattern variable.

We also speak a bit loosely of pred alone as a pattern predicate, and mean by that the collection of all pattern predicates obtained for different parameter values *params*. We say that m is a *monotonic* predicate, if for all possible parameter values *params* and all data sets $\mathcal{D}$:

$$\forall\ \phi,\psi \in \mathcal{L}\ \text{ such that } \phi \succeq \psi : \mathsf{m}(\psi,\mathcal{D},params) \rightarrow \mathsf{m}(\phi,\mathcal{D},params)$$

The class of *anti-monotonic* predicates is defined dually. Thus, minfreq, ismoregeneral, and length_atmost are monotonic, their duals are anti-monotonic.

A pattern predicate $\mathsf{pred}(\phi,\mathcal{D},params)$ defines the *solution set* $Th(\mathsf{pred}(\phi,\mathcal{D},params),\mathcal{L}) = \{\psi \in \mathcal{L} \mid \mathsf{pred}(\psi,\mathcal{D},params) = true\}$. Furthermore, for monotonic predicates m these sets will be monotone, i.e. for all $\phi \succeq \psi \in \mathcal{L} : \psi \in Th(\mathsf{m},\mathcal{L}) \rightarrow \phi \in Th(\mathsf{m},\mathcal{L})$.

Example 2. Let $\mathcal{L} = \mathcal{L}_\Sigma$ with $\Sigma = \{\mathtt{a},\mathtt{c},\mathtt{g},\mathtt{t}\}$ and $\mathcal{D} = \{\mathtt{acca},\mathtt{at},\mathtt{gg},\mathtt{cga},\mathtt{accag},\mathtt{at},\mathtt{g}\}$. Then the following predicates evaluate to true: $\mathsf{minfreq}(\mathtt{acca};\ 2;\ \mathcal{D})$, $\mathsf{minfreq}(\mathtt{g};\ 4;\ \mathcal{D})$, $\mathsf{maxfreq}(\mathtt{ca};\ 2;\ \mathcal{D})$, $\mathsf{maxfreq}(\mathtt{t};\ 1;\ \mathcal{D})$.

The pattern predicate $Q_m := \mathsf{minfreq}(\phi,2,\mathcal{D})$ defines $Th(Q_m,\mathcal{L}_\Sigma) = \{\epsilon,\mathtt{a},\mathtt{c},\mathtt{g},\ \mathtt{ac},\mathtt{ca},\mathtt{cc},\mathtt{acc},\mathtt{cca},\mathtt{acca}\}$, and the pattern predicate $Q_a := \mathsf{maxfreq}(\phi,3,\mathcal{D})$ defines the infinite set $Th(Q_a,\mathcal{L}_\Sigma) = \mathcal{L}_\Sigma \setminus \{\epsilon,\mathtt{g}\}$.

The definition of $Th(\mathsf{pred}(\phi,\mathcal{D},params),\mathcal{L})$ is extended in the natural way to a definition of the solution set $Th(Q,\mathcal{L})$ for boolean combinations Q of pattern predicates over a unique pattern variable: $Th(\neg Q,\mathcal{L}) := \mathcal{L} \setminus Th(Q,\mathcal{L})$, $Th(Q_1 \vee Q_2,\mathcal{L}) := Th(Q_1,\mathcal{L}) \cup Th(Q_2,\mathcal{L})$. The predicates that appear in Q may reference one or more data sets $\mathcal{D}_1,\ldots,\mathcal{D}_n$.

We are interested in computing solution sets $Th(Q, \mathcal{D}, \mathcal{L})$ for boolean queries Q that are constructed from monotonic and anti-monotonic pattern predicates. De Raedt et al. [6] presented a technique to rewrite an arbitrary boolean query Q into an equivalent query of the form $Q_1 \vee ... \vee Q_k$ such that k is minimal and each of the subqueries Q_i is the conjunction $Q_{a_i} \wedge Q_{m_i}$ of a monotonic and an anti-monotonic query. This type of subquery is called *conjunctive*. De Raedt et al. argued that this is useful because 1) there exist effective algorithms for computing the solution space to such queries Q_i (cf. [5, 4, 6, 17] and below), and 2) that minimizing k in this context would also minimize the number of calls to such algorithms and thus corresponds to a kind of inductive query optimization. In the present paper, we focus on the subproblem of optimizing conjunctive inductive queries, which is an essential step in this process. Observe that in a conjunctive query $Q_a \wedge Q_m$, Q_a and Q_m need not be atomic expressions. Indeed, it is well-known that both the disjunction and conjunction of two monotonic (resp. anti-monotonic) predicates are monotonic (resp. anti-monotonic). Furthermore, the negation of a monotonic predicate is anti-monotonic and vice versa.

We will assume that there are cost-functions c_a and c_m associated to the anti-monotonic and monotonic subqueries Q_a and Q_m. The idea is that the cost functions reflect the (expected) costs of evaluating the query on a pattern. E.g., $c_a(\phi)$ denotes the expected cost needed to evaluate the anti-monotonic query Q_a on the pattern ϕ. The present paper will neither propose realistic cost functions nor address the nature of these cost functions. Even though it is clear that some predicates are more expensive than other ones, more work seems needed in order to obtain cost estimates that are as reliable as in traditional databases. The present use of cost-functions is only a first step in this direction. One point to mention is also that several of the traditional pattern mining algorithms, such as Agrawal et al.'s Apriori [2] and the levelwise algorithm [18], try to minimize the number of passes through the data. Even though this could also be cast within the present framework, the cost functions introduced above better fit the situation where the data can be stored in main memory. One direct application would concern molecular feature mining [17, 16], where one aims at discovering fragments (i.e. subgraphs) within molecules (represented as graph structures). In this case, a natural cost function is to minimize the number of covers tests (i.e. matching a pattern with a molecule or graph) because each covers test corresponds to a subgraph isomorphism problem, a known NP-complete problem.

By now, we are able to formulate the *conjunctive inductive query optimization problem* that is addressed in this paper:

Given
- a language $\mathcal{L}$ of patterns,
- a conjunctive query $Q = Q_a \wedge Q_m$
- two cost functions c_a and c_m from $\mathcal{L}$ to $\mathbb{R}$

Find the set of patterns $Th(Q, \mathcal{D}, \mathcal{L})$, i.e. the solution set of the query Q in the language $\mathcal{L}$ with respect to the database $\mathcal{D}$, in such a way that that the total cost needed to evaluate patterns is as small as possible.

One useful property of conjunctive inductive queries is that their solution space $Th(Q, \mathcal{D}, \mathcal{L})$ is a version space (sometimes also called a convex space).

Definition 3. *Let $\mathcal{L}$ be a pattern language, and $I \subseteq \mathcal{L}$. I is a version space, if $\forall \phi, \phi', \psi \in \mathcal{L} : \phi \preceq \psi \preceq \phi'$ and $\phi, \phi' \in I \implies \psi \in I$.*

Version spaces are particularly useful when they can be represented by boundary sets, i.e. by the sets $G(Q, \mathcal{D}, \mathcal{L})$ of their maximally general elements, and $S(Q, \mathcal{D}, \mathcal{L})$ of their minimally general elements. Finite version spaces are always boundary set representable, cf. [14], and this is what we will assume from now on.

Example 4. Continuing from Ex. 2, let $Q = Q_m \wedge Q_a$. We have $Th(Q, \mathcal{L}_\Sigma, \mathcal{D}) = \{\texttt{a}, \texttt{c}, \texttt{ac}, \texttt{ca}, \texttt{cc}, \texttt{acc}, \texttt{cca}, \texttt{acca}\}$. This set of solutions is completely characterized by $S(Q, \mathcal{L}_\Sigma, \mathcal{D}) = \{\texttt{acca}\}$ and $G(Q, \mathcal{L}_\Sigma, \mathcal{D}) = \{\texttt{a}, \texttt{c}\}$.

3 Solution Methods

In this section, we first introduce a data structure, called the partial version space tree, and then show how it can be used for addressing the optimization problem.

3.1 Partial version space trees

A partial version space tree is an extension of of a *suffix trie*. This is a tree $\mathcal{T}$ with the following properties:

- Each edge is labelled with a symbol from Σ, and all outgoing edges from a node have different labels.
- Each node $n \in \mathcal{T}$ uniquely represents a string $s(n) \in \Sigma^\star$ which is the concatenation of all labels on the path from the root to n. We define $s(root) = \epsilon$. If it is clear from the context, we will simply write n instead of $s(n)$.
- For each node $n \in \mathcal{T}$ there is also a node $n' \in \mathcal{T}$ for all suffixes n' of n. Furthermore, if $n \neq root$, there is a *suffix-link* to the longest proper suffix of n, which we denote by $\mathit{suffix}(n)$. We write $\mathit{suffix}^2(n)$ for $\mathit{suffix}(\mathit{suffix}(n))$ etc. and define $\mathit{suffix}^i(root) = \perp \ \ \forall\, i \in \mathbb{N}$, where $\perp$ is a unique entity.

To obtain a partial version space tree, we augment each node $n \in \mathcal{T}$ with the following information:

- There are two different labels l_m and l_a, one for the monotonic and one for the anti-monotonic constraint. Each label may obtain one of the values $\oplus$ or $\ominus$, indicating that the string $s(n)$ satisfies the constraint or not. If the truth value of a constraint has not been determined yet, the corresponding label gets a $?$.
- There is a link to the father of n, denoted by $\mathit{parent}(n)$. For example, with $n = \texttt{abc}$ we have $\mathit{parent}(n) = \texttt{ab}$. As for $\mathit{suffix}(n)$ we write $\mathit{parent}^2(n)$ for $\mathit{parent}(\mathit{parent}(n))$ etc. and define $\mathit{parent}^i(root) = \perp \ \ \forall\, i \in \mathbb{N}$.

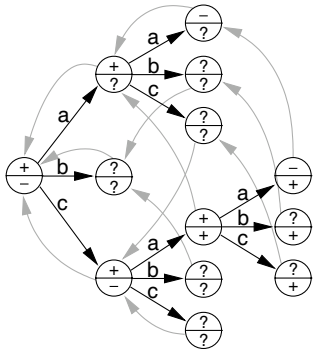


Fig. 1. An example of a partial version space tree. Here and in the rest of this paper, suffix-links are drawn lighter to distinguish them from the black parent-to-child links.

– There is a list of links to all incoming suffix-links to n which we denote by $isl(n)$. For example, if $n =$ `ab` and `aab`, `cab` are the only nodes in $\mathcal{T}$ that have n as their suffix, $isl(n) = \{\texttt{aab}, \texttt{cab}\}$.

Furthermore, the following conditions are imposed on partial version space trees:

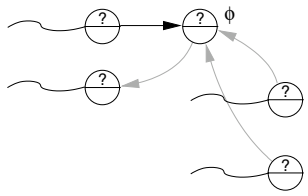
(C1) for all leaves n in $\mathcal{T}$, either $l_m(n) = \ominus$ or $l_m(n) = \textcircled{?}$, and all nodes with $l_m = \ominus$ are leaves.
(C2) all expanded nodes n have $l_m(n) = \oplus$

The first condition is motivated by the monotonicity of our query Q_m: if n does not satisfy Q_m, none of its descendants can satisfy the monotonic constraint either, so they need neither be considered nor expanded. A consequence of these requirements is that nodes n with $l_m(n) = \oplus$ must always be expanded. An example of a partial version space tree can be seen in Fig. 1, where the upper part of a node stands for the monotonic and the lower part for the anti-monotonic label.

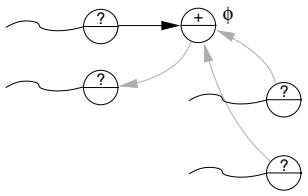
The algorithm given in the next subsection computes the version space tree starting from the tree containing only the root and then expanding the tree until no $\textcircled{?}$-labels are found. Then, the solution to the original query consists of all nodes n in $\mathcal{T}$ that have a $\oplus$ in both of their labels, i.e. they are of type $\oplus\!\!\!\!\oplus$. As described in [6], it is also possible to construct the boundary sets S and G from the partial version space tree.

3.2 An Algorithmic Framework

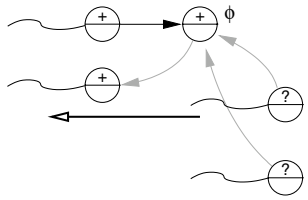
We are now ready to present an algorithmic framework for computing $Th(Q_m \wedge Q_a, \mathcal{D}, \mathcal{L}_\Sigma)$. The key idea is that instead of constructing the version space tree in a top-down, Apriori-like manner, we allow for more freedom in selecting the pattern ϕ and the query $Q_a(\phi)$ or $Q_m(\phi)$ to be evaluated. By doing so, we



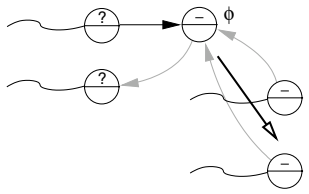
(a) Node ϕ is tested against Q_m. Note that ϕ can't have any children due to condition (C2).



(b) If it is positive, we mark it accordingly...



(c) ... and propagate this label up to the root until we reach a node that has already been marked positive.



(d) If it is negative, the label is propagated down to the leaves by recursively following the incoming suffix-links.

Fig. 2. How monotone labels are propagated in the tree.

hope to decrease the total cost of obtaining the solution space. As a motivating example, assume our alphabet is $\Sigma = \{\texttt{a}, \texttt{b}, \texttt{c}\}$ and pattern $\phi = \texttt{abc}$ turns out to satisfy Q_m. Then, by the monotonicity of Q_m, we know that all patterns more general than ϕ satisfy Q_m as well, so ϵ, a, b, c, ab and bc need no longer be tested against Q_m. Thus, by evaluating ϕ, we also obtain the truth values (w.r.t. Q_m) of six other patterns, which would all have been tested using a levelwise strategy. If, on the other hand, ϕ does not satisfy Q_m, we know that all patterns more specific than ϕ cannot satisfy Q_m, so the node representing ϕ need not be expanded. (This last observation corresponds to the Apriori pruning-strategy.)

This suggests the following approach: Whenever a pattern ϕ is positive w.r.t. Q_m, we propagate the monotonic $\oplus$-label up to the root by recursively following ϕ's parent- and suffix-links, until we reach a node that has already been marked positive. Furthermore, ϕ will be expanded and all of its children are labelled appropriately. If ϕ does not satisfy Q_m, we stop the expansion of this node and propagate the monotonic $\ominus$ down to the leaves by following the incoming suffix-links of ϕ. See Fig. 2 for a schematic overview of these operations.

For the anti-monotonic query Q_a, we propagate the labels in opposite directions. That is, a $\oplus$ is propagated down to the leaves (by following the children-

and incoming suffix-links) and a $\ominus$ up to the root. The corresponding algorithm is shown in Fig. 3. We use a priority queue P to store those nodes whose truth value has not been fully determined, i.e. all nodes of types $\oplus$, $\dot{\oplus}$, $\dot{\ominus}$ and $\dot{?}$. The queue not only returns the next pattern ϕ to be evaluated, but also a variable *pred* that tells us which of the predicates Q_m or Q_a should be evaluated for ϕ. Whenever a change of labels results in a label of the other type ($\dot{\oplus}$, $\dot{\ominus}$, $\dot{?}$, $\ominus$ or $\oplus$), we remove the node from P. Note that nodes of type $\dot{?}$ are also deleted from P although their anti-monotonic part is undetermined; cf. the above discussion.

Input: a query $Q = Q_m \wedge Q_a$ and a database $\mathcal{D}$
Output: a version space tree $\mathcal{T}$ representing $Th(Q, \mathcal{D}, \mathcal{L})$
VSTree $\mathcal{T} \leftarrow \{(\epsilon, \dot{?})\}$ // insert empty string and mark it as unseen
PriorityQueue $P \leftarrow \{\epsilon\}$
while $(|P| > 0)$
  $(\phi, pred) \leftarrow P$.next // get next pattern and predicate to be evaluated

  if $(pred = antimonotone)$ // evaluate Q_a by accessing $\mathcal{D}$
    if $(Q_a(\phi, \mathcal{D}))$ // a pattern satisfying Q_a
      propagate $\dot{\ominus}$ down to the leaves and remove determined patterns from P
    else // a pattern not satisfying Q_a
      propagate $\ominus$ up to the root and remove determined patterns from P

  else if $(pred = monotone)$ // evaluate Q_m by accessing $\mathcal{D}$
    if $(Q_m(\phi, \mathcal{D}))$ // a pattern satisfying Q_m
      propagate $\dot{\oplus}$ up to the root and remove determined patterns from P
      expand ϕ in $\mathcal{T}$
      for all children ψ of ϕ // set children's labels
        if $(l_m(\mathit{suffix}(\psi)) = \ominus)$ $l_m(\psi) \leftarrow \ominus$ else $l_m(\psi) \leftarrow \dot{?}$
        if $(l_a(\phi) = \oplus$ or $l_a(\mathit{suffix}(\psi)) = \oplus)$ $l_a(\psi) \leftarrow \oplus$ else $l_a(\psi) \leftarrow \dot{?}$
        insert ψ in P if it is not fully determined
    else // a pattern not satisfying Q_m
      propagate $\dot{\ominus}$ down to the leaves and remove determined patterns from P
return $\mathcal{T}$

Fig. 3. An algorithmic framework

The choice of priorities for nodes determines the search strategy being used. By assigning the highest priorities to the most shallow nodes (i.e. nodes that are close to the root), a level wise search is simulated as in [6]. On the other hand, by assigning the highest priorities to the deepest nodes, we are close to the idea of *Dualize & Advance* [10, 9], since we will go deeper and deeper into the tree until we encounter a node that has only negative children. Somewhere in the middle between these two extremes lies a completely randomized strategy, which assigns random priorities to all nodes. In the next subsection, we propose an algorithm that tries to approximate an *optimal* strategy.

3.3 Towards An Optimal Strategy

Let us first assign four counters to each node ϕ in the partial version space tree $\mathcal{T}$: $z_m(\phi)$, $z_{\neg m}(\phi)$, $z_a(\phi)$ and $z_{\neg a}(\phi)$. Each of them counts how many labels of nodes in $\mathcal{T}$ would be marked if ϕ's label were changed (including ϕ itself). For example, $z_m(\phi)$ counts how many monotone labels would be marked $\oplus$ if ϕ turned out to satisfy Q_m. In the tree in Fig. 1, we have $z_m(\texttt{cab}) = 3$, because marking `cab` with a $\ominus$ would result in marking `ab` and `b` as well, whereas $z_m(\texttt{ac}) = 1$, because no other nodes could be marked in their monotonic part. Likewise, $z_{\neg m}(\phi)$ counts how many monotone labels would change to $\ominus$ if $Q_m(\phi, \mathcal{D})$ turned out to be false. The $z_a(\phi)$- and $z_{\neg a}(\phi)$-counters form the anti-monotonic counterpart. We define $z_m(\phi) = z_{\neg m}(\phi) = 0$ if ϕ's monotonic label is $\neq$?, and likewise for z_a and $z_{\neg a}$. If there is no confusion about which node we talk, we will simply write z_m instead of $z_m(\phi)$ etc.

Assume further that we know the following values for each pattern:

- $P_m(\phi, \mathcal{D})$, the probability that ϕ satisfies the monotonic predicate Q_m in database $\mathcal{D}$,
- $P_a(\phi, \mathcal{D})$, the dual of P_m for the anti-monotonic predicate Q_a,
- $c_m(\phi, \mathcal{D})$, the costs for evaluating the monotonic predicate $Q_m(\phi, \mathcal{D})$ and
- $c_a(\phi, \mathcal{D})$, the dual of c_m for the anti-monotonic predicate Q_a.

Now

$$P_m(\phi, \mathcal{D}) \cdot z_m(\phi) + (1 - P_m(\phi, \mathcal{D})) \cdot z_{\neg m}(\phi)$$

is the expected value of the number of monotone labels that get marked by evaluating Q_m for pattern ϕ. Since the operation of evaluating $Q_m(\phi, \mathcal{D})$ has costs $c_m(\phi, \mathcal{D})$, we see that the average number of marked labels per cost unit are

$$\frac{1}{c_m(\phi, \mathcal{D})}(P_m(\phi, \mathcal{D}) \cdot z_m(\phi) + (1 - P_m(\phi, \mathcal{D})) \cdot z_{\neg m}(\phi)).$$

A similar formula holds for the average number of marked anti-monotone labels, so the optimal node in the partial version space tree is the one where

$$\max\{\tfrac{1}{c_m}(P_m z_m + (1 - P_m) \cdot z_{\neg m}), \tfrac{1}{c_a} \cdot (P_a z_a + (1 - P_a) \cdot z_{\neg a})\} \qquad (1)$$

is maximal.

The question now is how to determine the probabilities and costs. For certain types of queries and databases it is conceivable that costs grow with increasing complexity of the patterns. For example, testing the coverage of a string becomes more expensive as the pattern becomes longer. On the other hand, short patterns are more likely to satisfy Q_m than long ones (and vice versa for Q_a). Therefore, length could be taken into account when approximating the the above costs and probabilities, but let us for now assume that there is no prior knowledge about those values, so we simply take $P_m = P_a = \frac{1}{2}$ and $c_m = c_a = 1$. With uniform costs and probabilities (1) breaks down to $\frac{1}{2}\max\{z_m + z_{\neg m}, z_a + z_{\neg a})\}$, where we can drop the constant factor $\frac{1}{2}$ because it has no impact on the relative order of the nodes.

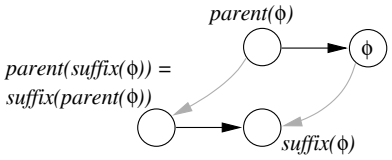


Fig. 4. Calculating z_m: When summing up $z_m(\mathit{parent}(\phi))$ and $z_m(\mathit{suffix}(\phi))$, $z_m(\mathit{parent}(\mathit{suffix}(\phi)))$ has been counted twice.

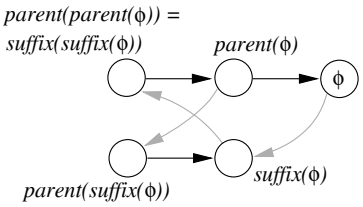


Fig. 5. If $\phi = \gamma_1 \dots \gamma_n$ ($\gamma_i \in \Sigma$) and $\mathit{suffix}^2(\phi) = \mathit{parent}^2(\phi)$, we have $\gamma_1 \dots \gamma_{n-3} = \gamma_3 \dots \gamma_n$, i.e. $\gamma_1 = \gamma_3 = \gamma_5 = \dots, \gamma_2 = \gamma_4 = \dots$

3.4 Calculating the Counter-Values

Next, we show how the four counter-values can be computed. Let us start with the $z_m(\phi)$-counter. Since a monotone $\oplus$-label for ϕ would be propagated up to the root by following ϕ's parent- and suffix-link, we basically have that $z_m(\phi)$ is the sum of $z_m(\mathit{parent}(\phi))$ and $z_m(\mathit{suffix}(\phi))$ plus 1 for ϕ itself. But, due to the fact that $\mathit{parent}(\mathit{suffix}(\phi)) = \mathit{suffix}(\mathit{parent}(\phi))$, we have that this z_m-value has been counted twice; we thus need to subtract it once (see Fig. 4):

$$z_m(\phi) \approx z_m(\mathit{parent}(\phi)) + z_m(\mathit{suffix}(\phi)) - z_m(\mathit{parent}(\mathit{suffix}(\phi))) + 1 \quad (2)$$

There are some exceptions to this equation: The easiest case is when $\mathit{suffix}(\phi) = \mathit{parent}(\phi)$, which happens if $\phi = \gamma^n$ for $\gamma \in \Sigma, n \in \mathbb{N}$. We then have $z_m(\phi) = z_m(\mathit{parent}(\phi)) + 1$ because $\mathit{parent}(\phi)$ is the only immediate generalization of ϕ. A slightly more complicated exception is when $\mathit{suffix}^2(\phi) = \mathit{parent}^2(\phi)$, which happens when $\phi = \gamma\delta\gamma\delta\gamma\dots$ for $\gamma, \delta \in \Sigma$ (see Fig. 5). Then $z_m(\phi) = z_m(\mathit{parent}(\phi)) + 2$, because all patterns that are more general than ϕ (apart from $\mathit{suffix}(\phi)$) are already counted by $z_m(\mathit{parent}(\phi))$. Similar rules for calculating z_m hold for exceptions of the type $\mathit{suffix}^3(\phi) = \mathit{parent}^3(\phi)$ etc. We summarize these results in the following

Lemma 5. *The counter-value for z_m is given by*

$$z_m(\phi) = \begin{cases} z_m(\mathit{parent}(\phi)) + n & \text{if } \mathit{suffix}^n(\phi) = \mathit{parent}^n(\phi) \\ z_m(\mathit{parent}(\phi)) + z_m(\mathit{suffix}(\phi)) - \\ \quad z_m(\mathit{parent}(\mathit{suffix}(\phi))) + 1 & \text{otherwise} \end{cases}$$

where we have to take the smallest value of n for which the "exceptional" case applies.

In practical implementations of this method it is advisable to "cut off" the search for the exceptional cases at a fixed depth (e.g. up to $\mathit{suffix}^2(\phi) = \mathit{parent}^2(\phi)$) and take the value of the "otherwise"-case as an approximation to the true value of z_m.

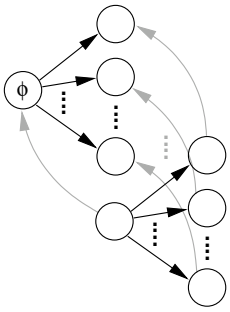


Fig. 6. Calculating z_a: When summing up z_a of ϕ's children and incoming suffix-links, the z_a's of the children of the incoming suffix-links have been counted twice.

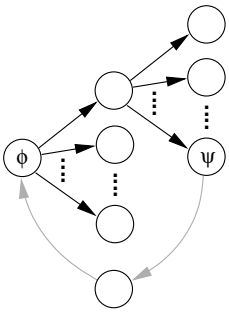


Fig. 7. One of ϕ's grandchildren (ψ) has one of ϕ's incoming suffix-links as its own suffix. So ψ has been counted twice and needs to be subtracted once.

Since anti-monotone $\ominus$-labels are propagated in the same direction as monotone $\oplus$-labels, lemma 5 holds for $z_{\neg a}$ as well. For the remaining two counters, we have to "peek" in the other direction, i.e. we have to consider the children and incoming suffix-links of ϕ. In a similar manner as we did for z_m, we have to consider the values that have been counted twice when summing over the z_a's of ϕ's children and incoming suffix-links. These are exactly the children of all incoming suffix-links, because their suffix-links point to the children of ϕ (see Fig. 6). We thus get the basic formula

$$z_a(\phi) \approx \sum_{\psi \in children(\phi)} z_a(\psi) + \sum_{\psi \in isl(\phi)} z_a(\psi) - \sum_{\psi \in children(isl(\phi))} z_a(\psi) + 1. \qquad (3)$$

Again, we need to consider some special cases where the above formula does not hold. The first is when one of ϕ's children has ϕ as its suffix, which happens iff $\phi = \gamma^n$ for $\gamma \in \Sigma, n \in \mathbb{N}$, because one of ϕ's sons is γ^{n+1}. In this case, we just sum *once* over this node and do *not* subtract z_a of γ^{n+1}'s children, because they were counted only once. The second exception arises when one of ϕ's grandchildren, say ψ, has one of ϕ's incoming suffix-links as its suffix, see Fig. 7. Again, this happens when $\phi = \gamma\delta\gamma\delta\gamma\ldots$ for $\gamma, \delta \in \Sigma$ and $\psi = \phi\gamma\delta$. Then a 1 for ψ needs to be subtracted from (3), because it has been counted twice. Note that the z_a's of ψ's children have not been counted twice; they have already been subtracted by the last sum in (3). There are many more exceptions, and the correct formula for the update is relatively complicated; as we only experimented with an approximation, we just give the equation that captures the above two cases (*children* is abbreviated as *ch*):

$$z_a(\phi) \approx \sum_{\psi \in ch(\phi)} z_a(\psi) + \sum_{\substack{\psi \in isl(\phi) \\ \psi \notin ch(\phi)}} \Big(z_a(\psi) - \sum_{\chi \in ch(\psi)} z_a(\chi) \Big) \; (+1) \qquad (4)$$

Here, the 1 is only added if the exception from Fig. 7 does not arise. Note that the equations for z_m and z_a have a recursive structure, because for calculating a counter-value we employ the corresponding counter-values of other nodes. So care has to be taken in which direction we traverse the tree when updating these values. In general, we work in the opposite directions than we would do if we propagated the corresponding label in the tree. Take, for example, the case where ϕ's monotonic label changes from ⍰ to $\ominus$. Whereas we propagate this label down to the leaves, we then need to adjust the $z_{\neg m}$-counters of all nodes between ϕ and the root. So we first adjust $z_{\neg m}$ of ϕ's father and suffix, then proceed to their fathers and suffices, etc. Again, the recursion stops when we reach a node that has already been marked in its monotonic part (and thus has $z_m = z_{\neg m} = 0$).

4 Experimental Results

We implemented the algorithm from Fig. 3 with two different queuing-strategies. The first, called *Random*, uses random priorities for the nodes in queue P. The second strategy, called *CounterMax*, works with the counter-values from Sect. 3.3, where we chose uniform costs and probabilities. We checked for exceptional cases up to *suffix*^{2}, as explained in Sect. 3.4. According to the algorithm and to (1), each pattern is tested either against Q_m or Q_a, depending on which of the subquery yields the maximum. We compared the results to an implementation of algorithm VST which constructs the version space tree in two passes (called DESCEND and ASCEND). The DESCEND algorithm is a straightforward adaptation of the Apriori and levelwise algorithm for use with strings and version space trees. It computes the set of all solutions w.r.t. Q_m. ASCEND starts from this result working bottom up and starting from the leaves of the tree. For each leaf, ASCEND tests whether it satisfies Q_a, if it does, the parent of the leaf will be tested; if it does not, the pattern is labelled $\ominus$ and the labels are propagated towards the parents and suffixes, more details can be found in [6].

We used a nucleotide database to compare the three algorithms, so our alphabet was $\Sigma = \{\mathtt{a}, \mathtt{c}, \mathtt{g}, \mathtt{t}\}$. The first dataset $\mathcal{D}_1$ was used for a minfrequency query and consisted of the first hundred nucleotide sequences from the *Hepatitis C* virus of the NIH genetic sequence database *GenBank* [22]. The second dataset $\mathcal{D}_2$ held the first hundred sequences from the *Escherichia coli* bacterium and was used for a maxfrequency query. The average length of the entries in $\mathcal{D}_1$ was about 500, the maximum 10,000. For $\mathcal{D}_2$ we had the values 2,500 and 30,000, respectively. We do not pretend that our results have any biological relevance; we simply used these datasets as a testbed for the different methods. We ran each algorithm several times for the query

$$\mathsf{minfreq}(\phi; min; \mathcal{D}_1) \wedge \mathsf{maxfreq}(\phi; max; \mathcal{D}_2),$$

where each of the variables min and max could take one of the values $\{2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20, \ldots, 95\}$. Some of the results can be seen in Fig. 8.

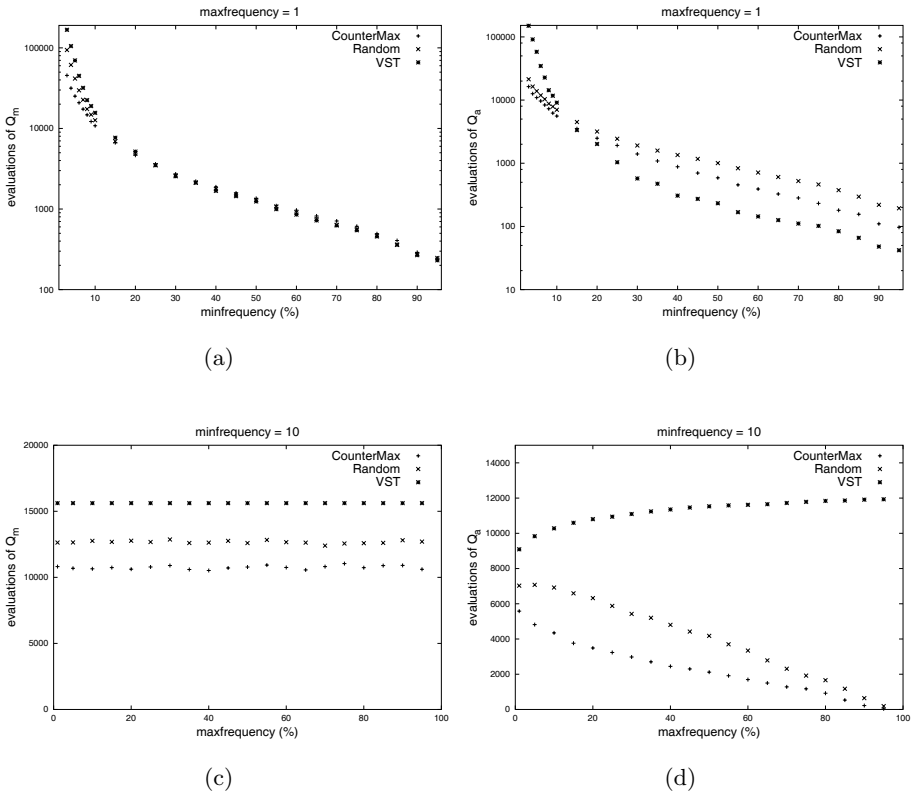


Fig. 8. A comparison of the three algorithms. Note that in figures (a) and (b) a logarithmic scale has been used.

Figures (a) and (b) show how the number of database accesses grows with decreasing values for *min* when the maximum frequency is constant. Although *max* has been fixed to 1, similar graphs could be shown for other values of *max*. Note that in the region where the hard instances lie ($min \in \{2, \ldots, 10\}$), *CounterMax* performs significantly better than VST. This is in particular true for the number of evaluations of Q_a (Fig. (b)). For the easy instances, our method takes slightly more evaluations of both predicates. This is obvious; if the length of the longest pattern satisfying Q_m is small, it is very unlikely to beat a levelwise method.[1] The random strategy lies somewhere in the middle between those two other methods.

[1] This fact has already been pointed out by [10]. We double-checked *CounterMax* on a different database (containing ORFs from yeast) where no performance improvement compared to VST could be seen either.

Figures (c) and (d) show the performance when *min* is fixed and *max* changes. The first thing to note is that in Fig. (c) the number of evaluations of Q_m is constant for VST. This is a simple consequence of how the algorithm works. Again, *CounterMax* takes less evaluations than VST, and *Random* is in between. In Fig. (d) we can see that for VST, the number of evaluations of Q_a levels off when *max* decreases, whereas the other two methods behave conversely. The reasons for this are clear: VST treats the anti monotonic query in a bottom-up manner by starting at the leaves. When it encounters a negative pattern w.r.t. Q_a, it propagates this label up to the root. This is more likely to happen at the leaves for small *max*, so in these cases it saves a lot of evaluations of the anti monotonic predicate. For methods *CounterMax* and *Random* it is better when positive patterns (w.r.t. Q_a) are close to the root, which happens for large *max*, because then all newly expanded nodes will "automatically" be marked with a $\ominus$ and need never be tested against Q_a.

5 Related Work and Conclusions

Algorithms that try to minimize the total number of predicate evaluations have been around for several years, most notably Gunopulos et al.'s *Dualize & Advance*-algorithm [10, 9] that computes $S(\mathsf{minfreq}(\phi, \cdot, \mathcal{D}), \mathcal{L})$ in the domain of itemsets. This works roughly as follows: first, a set *MS* of maximal specific sentences is computed by a randomized depth-first search. Then the negative border of *MS* is constructed by calculating a *minimum hypergraph transversal* of the complements of all itemsets in *MS*. This process is repeated with the elements from the hypergraph transversal until no more maximal specific sentences can be found. The result is then set of *all* maximal interesting itemsets.

Although Gunopulos et al. work with itemsets and only consider monotonic predicates, there is a clear relation to our approach. Whereas the former method needs to compute the minimum hypergraph transversals to find the candidates for new maximal interesting sentences, these can be directly read off the partial version space tree. In fact, all nodes whose monotonic part is still undetermined are the only possible patterns of $S(\mathsf{minfreq}(\phi, \cdot, \mathcal{D}), \mathcal{L})$ that have not been found so far. These are exactly the nodes that are still waiting in the priority queue. So by performing a depth-first expansion until all children are negative, our algorithm's behaviour is close to that of *Dualize & Advance*. It should be noted that the two strategies are not entirely equal: if a node ϕ has negative children only, it is not necessarily a member of S because there could still be more specific patterns that have ϕ as their *suffix* and satisfy Q_m.

One of the fastest algorithms for mining maximal frequent sets is Bayardo's *Max-Miner* [3]. This one uses a special set-enumeration technique to find large frequent itemsets before it considers any of their subsets. Although this is completely different from what we do at first sight, *CounterSum* also has a tendency to test long strings first because they will have higher z_m-values. By assigning higher values to P_m for long patterns this behaviour can even be enforced.

As mentioned in the introduction, the work is related — at least in spirit — to several proposals for minimizing the number of membership queries when learning concepts, cf. [20].

Finally, let us mention that the present work is a significant extension of that by [6] in that we have adapted and extended their version space tree and also shown how it can be used for optimizing the evaluation of conjunctive queries. The presented technique has also shown to be more cost effective than the ASCEND and DESCEND algorithms proposed by [6].

Nevertheless, there are several remaining questions for further research. They include an investigation of cost functions for inductive queries, methods for estimating or approximating the probability that a query will succeed on a given pattern, determining the complexity class of the query optimization problem, addressing query optimization for general boolean queries, and considering other types of pattern domains. The authors hope that the presented formalization and framework will turn out to be useful in answering these questions.

Acknowledgements

This work was partly supported by the European IST FET project cInQ. The authors are grateful to Manfred Jaeger, Sau Dan Lee and Heikki Mannila for contributing the theoretical framework on which the present work is based, to Amanda Clare and Ross King for the yeast database and to Sau Dan Lee and Kristian Kersting for feedback on this work.

References

1. R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. VLDB*, 1994.
2. R. Agrawal, T. Imielinski, A. Swami. Mining association rules between sets of items in large databases. In *Proc. SIGMOD*, pp. 207-216, 1993.
3. R. Bayardo. Efficiently mining long patterns from databases. In *Proc. SIGMOD*, 1998.
4. C. Bucila, J. Gehrke, D. Kifer, W. White. DualMiner: A dual pruning algorithm for itemsets with constraints. In *Proc. of SIGKDD*, 2002.
5. L. De Raedt, S. Kramer. The levelwise version space algorithm and its application to molecular fragment finding. In *Proc. IJCAI*, 2001.
6. L. De Raedt, M. Jäger, S. D. Lee, H. Mannila: A Theory of Inductive Query Answering. In *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM'02)*, 123–130, Maebashi (Japan), 2002.
7. L. De Raedt, A perspective on inductive databases. *SIGKDD Explorations*, Vol. 4 (2), 2002.
8. B. Goethals, J. Van den Bussche. On supporting interactive association rule mining. In *Proc. DAWAK*, LNCS Vol. 1874, Springer Verlag, 2000.
9. D. Gunopulos, H. Mannila, S. Saluja. Discovering All Most Specific Sentences by Randomized Algorithms. In *Proc. ICDT*, LNCS Vol. 1186, Springer Verlag, 1997.
10. D. Gunopulos, R. Khardon, H. Mannila, H. Toivonen: Data mining, Hypergraph Transversals, and Machine Learning. In *Proceedings of the 16th ACM Symposion on Principles of Database Systems (PODS)*, Tuscon (Arizona), 1997.

11. J. Han, Y. Fu, K. Koperski, W. Wang, and O. Zaiane. DMQL: A Data Mining Query Language for Relational Databases.In *Proc. SIGMOD'96 Workshop on Research Issues on Data Mining and Knowledge Discovery*, Montreal, Canada, June 1996.
12. J. Han, L. V. S. Lakshmanan, and R. T. Ng. Constraint-Based, Multidimensional Data Mining, *Computer*, Vol. 32(8), pp. 46-50, 1999.
13. J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *Proc. SIGMOD*, 2000.
14. H. Hirsh. Generalizing Version Spaces. *Machine Learning*, Vol. 17(1): 5-46 (1994).
15. H. Hirsh. Theoretical underpinnings of versionspaces. In *Proc. IJCAI*, 1991.
16. Akihiro Inokuchi, Takashi Washio, Hiroshi Motoda: Complete Mining of Frequent Patterns from Graphs: Mining Graph Data. Machine Learning 50(3): 321-354 (2003)
17. S. Kramer, L. De Raedt, C. Helma. Molecular Feature Mining in HIV Data. In *Proc. SIGKDD*, 2001.
18. H. Mannila and H. Toivonen. Levelwise search and borders of theories in knowledge discovery, *Data Mining and Knowledge Discovery*, Vol. 1, 1997.
19. T. Mitchell. Generalization as Search, *Artificial Intelligence*, Vol. 18 (2), pp. 203-226, 1980.
20. T. Mitchell. Machine Learning. McGraw Hill. 1997.
21. R. T. Ng, L. V.S. Lakshmanan, J. Han, and A. Pang. Exploratory mining and pruning optimizations of constrained associations rules. In *Proc. SIGMOD*, 1998.
22. *NIC Genetic Sequence Database.* Available at `www.ncbi.nlm.nih.gov/Genbank/GenbankOverview.html`
23. E. Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, 1995.
24. P. Weiner. Linear pattern matching algorithm. In *Proc. 14th IEEE Symposium on Switching and Automata Theory*, pages 1–11, 1973.

What You Store Is What You Get (extended abstract)

Floris Geerts, Bart Goethals, and Taneli Mielikäinen

HIIT Basic Research Unit
Department of Computer Science
University of Helsinki, Finland

Abstract. Recent studies emerged the need for representations of frequent itemsets that allow to estimate supports. Several methods have been proposed that achieve this goal by generating only a subset of all frequent itemsets. In this paper, we propose another approach, that given a minimum support threshold, stores only a small portion of the original database from which the supports of frequent itemsets can be estimated. This representation is especially valuable in case one is interested in frequent itemset of size at least some threshold value. However, we also present methods for estimating the support of smaller frequent itemsets.

1 Introduction

Due to the large amount of frequent itemsets that can be generated from transactional databases, recent studies emerged the need for concise representations of all frequent itemsets. In the context of inductive databases, one wants to answer support queries as efficiently as possible. Not only the effort to obtain the supports is important, but also the storage capacity needed plays an important role. This resulted in several successful algorithms that generate only a small portion of all frequent itemsets that allow to derive the support of all other frequent itemsets [2,3,4,5,6,7]. The drawback of some of these algorithms is that they compute representations which are difficult to interpret and to understand.

Two extreme examples of such algorithms are the following: When the primary concern is the data storage, the database itself can serve as representation for the frequent sets. Indeed, in many cases the number of unique transactions in the database is smaller than the number of all frequent itemsets. To obtain the support of itemsets, one simply scans the database and count. When time is of importance, the set of frequent itemsets itself provides an efficient representation since the support of an itemset is obtained by a simple lookup.

It is therefore only natural to ask for intermediate representations that store parts of *both* the transaction database and the set of frequent itemsets. In this paper, we define such a representation and report some initial investigations. To the best of our knowledge, such intermediate representations have not been considered before.

The transaction database part of this representation is obtained by *trimming*. The idea is to modify the transaction database optimally for finding all frequent

sets of size larger than a certain size k. This is done by throwing away transactions from the database that do not contribute to the support of any frequent k-itemset and by throwing away items from all transactions that no longer occur in any frequent k-itemset supported by that transaction. A similar technique is used to speed up Apriori-like algorithms, such as DHP [11] and DCI [10].

Although the frequencies of frequent itemsets of size larger or equal to k can be retrieved easily from such a trimmed database, this is less obvious for smaller itemsets.

We provide three approaches to obtain information about the support of itemsets of size smaller than k. The first approach consists of simply adding these frequent sets to the representation; in this way, the representation becomes intermediate. The second approach estimates the support of an itemset by computing it relative the the trimmed database. The last approach uses affine transformations to obtain more accurate estimates for the supports of these small itemsets.

The paper is organized as follows. In Section 2 we provide the necessary definitions. Section 3 explains the process of database trimming and Section 4 shows three ways of dealing with small itemsets. In Section 5, we end the paper with some final thoughts and ideas for future work.

2 Preliminaries

Let $\mathcal{I}$ be a set of items. A subset of $\mathcal{I}$ is called an itemset, or a k-itemset if it consists of k items. A transaction over $\mathcal{I}$ is a pair (tid, I) where tid is the transaction identifier and I is an itemset. A transaction (tid, I) supports an itemset $J \subseteq \mathcal{I}$ if $J \subseteq I$. A transaction database $\mathcal{D}$ over $\mathcal{I}$ is a finite set of transaction over $\mathcal{I}$.

The cover of an itemset I in $\mathcal{D}$ is the set of transactions in $\mathcal{D}$ which support I:

$$\text{cover}\,(I, \mathcal{D}) := \{(tid, J) \in \mathcal{D} \mid I \subseteq J\}\,.$$

The support of an itemset I in $\mathcal{D}$, denoted by $\text{s}(I, \mathcal{D})$, is the size of its cover in $\mathcal{D}$. The cover of a set of itemsets $I_1, \ldots, I_k$ in $\mathcal{D}$, denoted by $\text{cover}(\{I_1, \ldots, I_k\}, \mathcal{D})$, is the union of the covers of the itemsets.

Let σ be a natural number, then an itemset I is σ-frequent in $\mathcal{D}$ if $\text{s}(I, \mathcal{D}) \geq \sigma$. We will denote the set of σ-frequent k-itemsets in $\mathcal{D}$ by $\mathcal{F}_{k,\sigma}(\mathcal{D})$, or $\mathcal{F}_{k,\sigma}$ when $\mathcal{D}$ is clear from the context. Also, $\mathcal{F}_\sigma := \cup_{i=0}^{\infty}\mathcal{F}_{i,\sigma}$, $\mathcal{F}_{\geq k,\sigma} := \cup_{i=k}^{\infty}\mathcal{F}_{i,\sigma}$ and $\mathcal{F}_{<k,\sigma} := \cup_{i=0}^{k-1}\mathcal{F}_{i,\sigma}$.

For any $(tid, I) \in \mathcal{D}$ and a set of itemsets $\mathcal{S}$, let

$$I\,[\mathcal{S}] := \{i \mid i \in J \wedge J \in \mathcal{S} \wedge J \subseteq I\}\,.$$

3 Trimming the database

In this section we propose a representation for all frequent itemsets of size at least a certain k. The representation consists of a transaction database which is smaller than the original transaction database, and is obtained by trimming the original database.

3.1 Horizontal Trimming

The horizontal trimming consists of throwing away transactions from the database that do not contribute to the support of any frequent k-itemset. Define for $k = 1, 2, \ldots, |\mathcal{I}|$,

$$\mathcal{H}_{k,\sigma} := \text{cover}(\mathcal{F}_{k,\sigma}, \mathcal{D}).$$

Since the number of transactions supporting frequent k-itemsets decreases as k increases. i.e.,

$$\mathcal{D} \supseteq \mathcal{H}_{1,\sigma} \supseteq \cdots \supseteq \mathcal{H}_{|\mathcal{I}|,\sigma},$$

we have that $\text{cover}(\mathcal{F}_{\geq k,\sigma}, \mathcal{D}) = \text{cover}(\mathcal{F}_{k,\sigma}, \mathcal{D})$. We obtain the following:

Lemma 1. *For any $k = 1, 2, \ldots, |\mathcal{I}|$, we have that for any $I \in \mathcal{F}_{\geq k,\sigma}$,*

$$\text{s}(I, \mathcal{D}) = \text{s}(I, \mathcal{H}_{k,\sigma}). \tag{1}$$

Proof. Since $\mathcal{H}_{k,\sigma} \subseteq \mathcal{D}$, we immediately have the inequality $\text{s}(I, \mathcal{D}) \geq \text{s}(I, \mathcal{H}_{k,\sigma})$. We now prove that for $I \in \mathcal{F}_{\geq k,\sigma}$ also $\text{s}(I, \mathcal{D}) \leq \text{s}(I, \mathcal{H}_{k,\sigma})$ holds. Let $(tid, J) \in \text{cover}(I, \mathcal{D})$. Now, $I \in \mathcal{F}_{\geq k,\sigma}$ implies that $\text{cover}(I, \mathcal{D}) \subseteq \text{cover}(\mathcal{F}_{\geq k,\sigma}, \mathcal{D}) = \mathcal{H}_{k,\sigma}$, and hence also $(tid, J) \in \text{cover}(I, \mathcal{H}_{k,\sigma})$. □

This lemma implies that itemsets of size at least k are σ-frequent in $\mathcal{D}$ if and only if they are frequent in $\mathcal{H}_{k,\sigma}$. Hence, the following theorem holds.

Theorem 1. *For any k and σ,*

$$\mathcal{F}_{\geq k,\sigma}(\mathcal{H}_{k,\sigma}) = \mathcal{F}_{\geq k,\sigma}(\mathcal{D}).$$

3.2 Vertical Trimming

The vertical trimming consists of throwing away items from all transactions that no longer occur in any frequent k-itemset supported by that transaction.

Define for $k = 1, 2, \ldots, |\mathcal{I}|$,

$$\mathcal{V}_{k,\sigma} := \{(tid, I[\mathcal{F}_{k,\sigma}]) \mid (tid, I) \in \mathcal{D}\}.$$

Similar to Theorem 1 we obtain:

Theorem 2. *For any k and σ,*

$$\mathcal{F}_{\geq k,\sigma}(\mathcal{V}_{k,\sigma}) = \mathcal{F}_{\geq k,\sigma}(\mathcal{D}).$$

3.3 Combined Trimming

The combined trimming consists of performing both horizontal and vertical trimming. Define for $k = 1, 2, \ldots, |\mathcal{I}|$,

$$\mathcal{D}_{k,\sigma} := \text{cover}(\mathcal{F}_{k,\sigma}, \mathcal{V}_{k,\sigma})$$

or equivalently,

$$\mathcal{D}_{k,\sigma} := \{(tid, I[\mathcal{F}_{k,\sigma}]) \mid (tid, I) \in \mathcal{H}_{k,\sigma}\}.$$

Again, we obtain:

Theorem 3. *For any k and σ,*

$$\mathcal{F}_{\geq k,\sigma}(\mathcal{D}_{k,\sigma}) = \mathcal{F}_{\geq k,\sigma}(\mathcal{D}).$$

Note that $\mathcal{D}_{k,\sigma}$ actually consists of all non-empty transactions in $\mathcal{V}_{k,\sigma}$.

3.4 An example

We illustrate the effect of trimming on the BMS-Webview-1 database [15]. The support is fixed to $\sigma = 36 = 0.06 \times 59\,602$. A lower support is not feasible due to the combinatorial explosion of the number of frequent itemsets. For $\sigma = 36$, the database contains 461 521 frequent itemsets.

The size of the trimmed database $\mathcal{D}_{k,\sigma}$ in function of k is shown in Figure 1. As can be seen, the size of the trimmed transaction database decreases very quickly when k increases.

Of course, in general this is not always the case. For example, the regularly used mushroom dataset from the UCI repository of machine learning databases [1] contains approximately 8 000 transactions of which almost all of them contain frequent itemsets of all sizes (for most widely used minimum support thresholds).

4 Taking care of the smaller itemsets

The database trimming we proposed in the previous section provides a concise representation for the frequent sets of size at least k. However, sometimes one is also interested in the small frequent sets. In this section, we propose three methods for estimating the support of itemsets of size smaller than k.

4.1 A naive lossless approach

A naive approach to obtain the exact support of any itemset of size smaller than k is to augment $\mathcal{D}_{k,\sigma}$ to a representation for all frequent sets by adding the frequent sets of size smaller than k.

Theorem 4. *For any k and σ,*

$$\mathcal{F}_{<k,\sigma}(\mathcal{D}) \cup \mathcal{F}_{\geq k,\sigma}(\mathcal{D}_{k,\sigma}) = \mathcal{F}_{\sigma}.$$

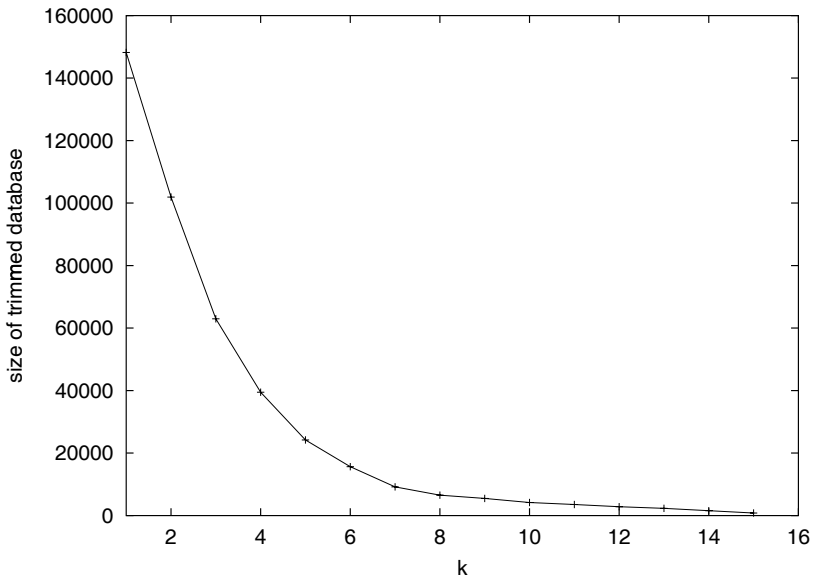


Fig. 1. Effect of trimming on BMS-Webview-1 database with $\sigma = 36$.

Note that for $k = 1$, the proposed representation actually consists of the original database from which all infrequent items and uncovered transactions are removed. The only frequent set stored in this case is the empty set whose support corresponds to the number of transactions in the original database. For other values of k, this representation consists of a part of the database and a part of all frequent itemsets.

In Figure 2 the representation for all frequent sets for the BMS-Webview-1 dataset and $\sigma = 36$ is shown. The figure shows for each k the size of this lossless representation. Additionally, the total size of the original database is shown, as well as the total size of all frequent itemsets. That is, the sum of the sizes of all transactions or itemsets respectively. As can be seen, for small k, the size of the proposed representation becomes half of the original database. For larger k, the trimmed database becomes much smaller (as could be seen on Figure 1), and hence, a lot of information contained in those transactions get lost, such that much more frequent itemsets need to be stored. Eventually, the other extreme is reached in which all frequent itemsets are stored and the trimmed database is empty. Note that for this dataset with the used minimum support threshold, the original database itself is already orders of magnitude smaller than the total size of all frequent itemsets. Hence, the space-time tradeoff of storing all frequent itemsets, or only storing the database becomes eminent in this case. Fortunately, intermediate solutions are near.

The complexity of retrieving the support of an arbitrary frequent itemset I then amounts to a simple lookup in $\mathcal{F}_{<k,\sigma}(\mathcal{D})$ if $|I| < k$. In the other case,

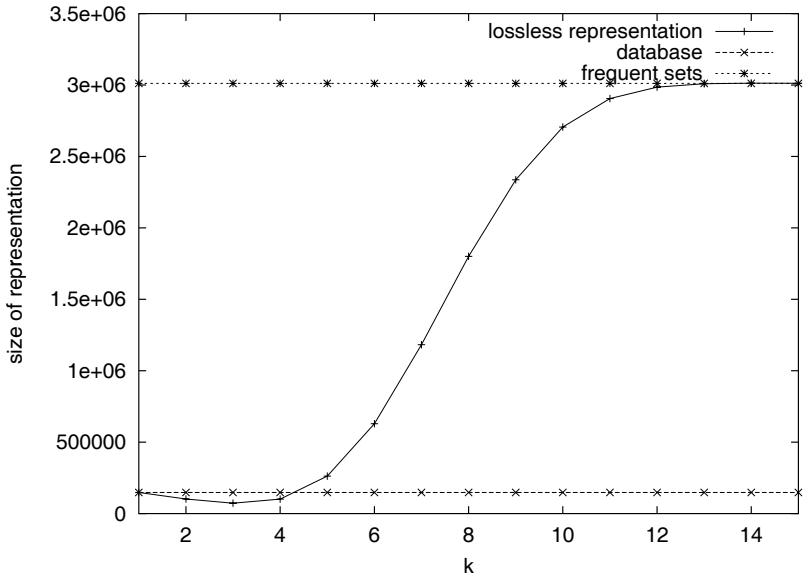


Fig. 2. The representation for $\mathcal{F}_\sigma$ for the BMS-Webview-1 dataset and $\sigma = 36$.

if $|I| \geq k$, we have to compute the support of I in the sometimes significantly smaller database $\mathcal{D}_{k,\sigma}$. Additionally, we can use any other concise representation on $\mathcal{F}_{<k,\sigma}(\mathcal{D})$ to reduce the total size even more [2,8,9,12,13].

4.2 A naive lossy approach

If the exact support of the itemsets of size smaller than k is not important, but an approximation suffices, then in many cases, $\mathcal{D}_{k,\sigma}$ itself can already provide a useful estimate for the support of these smaller itemsets.

In Figure 3 we show the squared error

$$\text{sqrerr}\left(\mathcal{F}_\sigma(\mathcal{D}), \mathcal{F}_\sigma(\mathcal{D}_{k,\sigma})\right) = \sum_{I \in \mathcal{F}_\sigma(\mathcal{D})} \left(\text{s}(I, \mathcal{D}) - \text{s}(I, \mathcal{D}_{k,\sigma})\right)^2$$

in function of k. Here, we used again the BMS-Webview-1 database with $\sigma = 36$.

4.3 A less lossy less naive approach

Another approach is to partition the frequent sets by grouping the frequent sets of size at most $k-1$ and to determine how each group should be corrected using one correction operator within one group. The grouping and correction operators for each group should have simple descriptions.

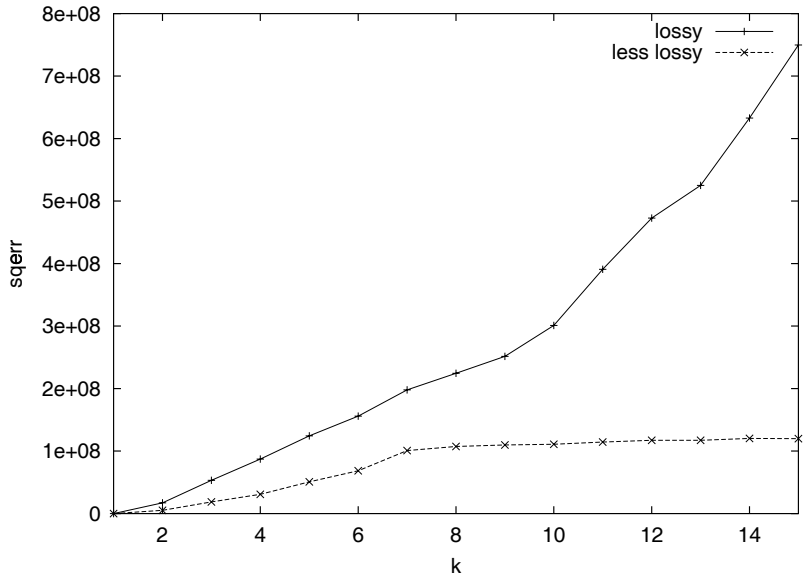


Fig. 3. The squared error for $a = 1$ and $b = 0$ compared with the squared error for optimal values for a and b for the BMS-Webview-1 database with $\sigma = 36$.

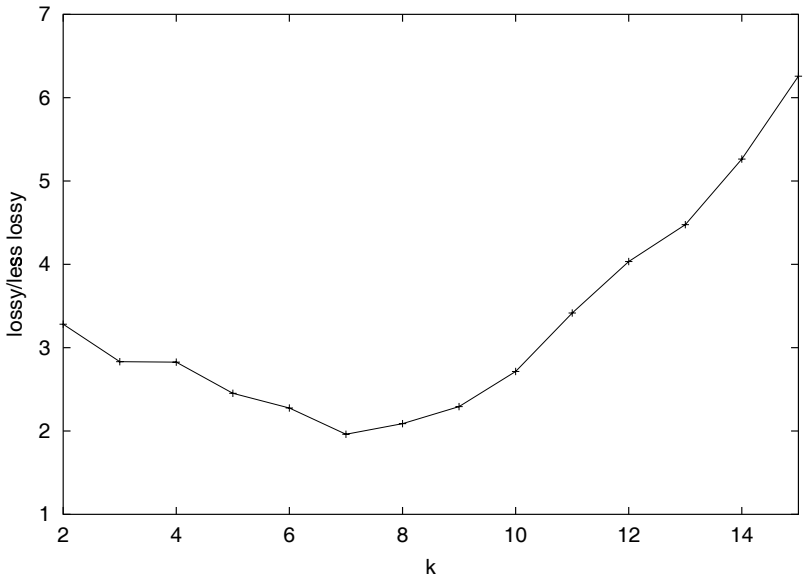


Fig. 4. The ratio of the squared error for $a = 1$ and $b = 0$ and the squared error for optimal values for a and b for the BMS-Webview-1 database with $\sigma = 36$.

As a concrete example, we consider the case where the grouping is determined by the size of the sets. That is, our groups that need to be corrected are $\mathcal{F}_{0,\sigma}, \ldots, \mathcal{F}_{k-1,\sigma}$. As a correction we compute an affine transformation $(x, a_i, b_i) \mapsto a_i x + b_i$ for each group $\mathcal{F}_{i,\sigma}, 0 \leq i \leq k-1$ that minimized the sum of squared differences between the correct support and the support estimated from $\mathcal{D}_k$:

$$\begin{aligned}\operatorname{sqrerr}\left(\mathcal{F}_\sigma(\mathcal{D}), \mathcal{F}_\sigma(\mathcal{D}_k)\right) &= \sum_{I \in \mathcal{F}_\sigma} \left(\mathrm{s}(I, \mathcal{D}) - (a_{|I|}\mathrm{s}(I, \mathcal{D}_{k,\sigma}) + b_{|I|})\right)^2 \\ &= \sum_{I \in \mathcal{F}_{<k,\sigma}} \left(\mathrm{s}(I, \mathcal{D}) - (a_{|I|}\mathrm{s}(I, \mathcal{D}_{k,\sigma}) + b_{|I|})\right)^2.\end{aligned}$$

The error sqrerr is minimized by choosing $(a_i, b_i), 0 \leq i \leq k-1$ to be (see e.g. [14]):

$$a_i = \frac{|\mathcal{F}_{i,\sigma}| \sum_{I \in \mathcal{F}_{i,\sigma}} \mathrm{s}(I, \mathcal{D})\mathrm{s}(I, \mathcal{D}_{k,\sigma}) - \left(\sum_{I \in \mathcal{F}_{i,\sigma}} \mathrm{s}(I, \mathcal{D})\right)\left(\sum_{I \in \mathcal{F}_{i,\sigma}} \mathrm{s}(I, \mathcal{D}_{k,\sigma})\right)}{|\mathcal{F}_{i,\sigma}| \sum_{I \in \mathcal{F}_{i,\sigma}} \mathrm{s}(I, \mathcal{D}_{k,\sigma})^2 - \left(\sum_{I \in \mathcal{F}_{i,\sigma}} \mathrm{s}(I, \mathcal{D}_{k,\sigma})\right)^2}$$

$$b_i = \frac{\sum_{I \in \mathcal{F}_{i,\sigma}} \mathrm{s}(I, \mathcal{D}) - a_i \sum_{I \in \mathcal{F}_{i,\sigma}} \mathrm{s}(I, \mathcal{D}_{k,\sigma})}{|\mathcal{F}_{i,\sigma}|}$$

For example, the correction (a_0, b_0) for the support of the empty set (i.e., all frequent sets of size 0 as $\mathcal{F}_{0,\sigma} = \{\emptyset\}$) is equal to $(|\mathcal{D}|/|\mathcal{D}_k|, 0)$. In general (a_i, b_i) attempts to tell how much the databases $\mathcal{D}$ and $\mathcal{D}_k$ differ when only considering the frequent i-itemsets: a_i scales the supports and b_i shifts them.

This method is illustrated in Figure 3, where we show the squared error for $a = 1$ and $b = 0$ compared with the squared error for optimal values for a and b for the BMS-Webview-1 database with $\sigma = 36$. To adjust the estimates even more, we also use the trivial information that supports are non-negative and cannot be larger than the number of transactions in the database. In Figure 4 we also compare the lossy approach with the less lossy approach by showing the ratio of the squared error for $a = 1$ and $b = 0$ and the squared error for optimal values for a and b.

5 Conclusions and future work

In the context of inductive databases for frequent itemsets, it is important to find good representations of the dataset in order to efficiently answer support queries. Instead of storing only a subset of all frequent itemsets, we propose to store parts of both the set of frequent itemsets and the database. In this way, the support of small frequent itemsets can be computed using a simple lookup, and the support of large frequent itemsets can be computed using a scan through a sometimes significantly reduced database.

When the representation is allowed to be lossy, two techniques are proposed which already show promising results with respect to the error on the support of an approximated itemset.

The proposed methodology of storing only a part of the database and a part of the frequent itemsets sheds a new light on representations, which we will investigate further. An interesting question is whether there exists an optimal separation of transactions and frequent itemsets, such that the resulting representation is small but still offers an efficient method to compute or approximate the support of all frequent itemsets.

Acknowledgements

We thank Blue Martini Software for contributing the KDD Cup 2000 data which we used in our experiments.

References

1. C.L. Blake and C.J. Merz. *UCI Repository of machine learning databases*. University of California, Irvine, Dept. of Information and Computer Sciences, `http://www.ics.uci.edu/~mlearn/MLRepository.html`, 1998.
2. J.-F. Bouliçaut and A. Bykowski. Frequent closures as a concise representation for binary data mining. In T. Terano, H. Liu, and A. L. P. Chen, editors, *Knowledge Discovery and Data Mining*, volume 1805 of *Lecture Notes in Artificial Intelligence*, pages 62–73. Springer-Verlag, 2000.
3. J.-F. Bouliçaut, A. Bykowski, and C. Rigotti. Free-sets: a condensed representation of Boolean data for the approximation of frequency queries. *Data Mining and Knowledge Discovery*, 7(1):5–22, 2003.
4. T. Calders and B. Goethals. Mining all non-derivable frequent itemsets. In T. Elomaa, H. Mannila, and H. Toivonen, editors, *Principles of Data Mining and Knowledge Discovery*, volume 2431 of *Lecture Notes in Artificial Intelligence*, pages 74–865. Springer-Verlag, 2002.
5. T. Calders and B. Goethals. Minimal k-free representations of frequent sets. In Nada Lavrac, Dragan Gamberger, Ljupco Todorovski, and Hendrik Blockeel, editors, *Proceedings of the 7th European Conference on Principles of Data Mining and Knowledge Discovery*, Lecture Notes in Artificial Intelligence. Springer-Verlag, 2003. To appear.
6. M. Kryszkiewicz. Concise representation of frequent patterns based on disjunction-free generators. In N. Cercone, T. Y. Lin, and X. Wu, editors, *Proceedings of the 2001 IEEE International Conference on Data Mining*, pages 305–312. IEEE Computer Society, 2001.
7. H. Mannila and H. Toivonen. Multiple uses of frequent sets and condensed representations. In E. Simoudis, J. Han, and U. M. Fayyad, editors, *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*, pages 189–194. AAAI Press, 1996.
8. T. Mielikäinen. Frequency-based views to pattern collections. In *IFIP/SIAM Workshop on Discrete Mathematics and Data Mining*, 2003.
9. T. Mielikäinen and H. Mannila. The pattern ordering problem. In Nada Lavrac, Dragan Gamberger, Ljupco Todorovski, and Hendrik Blockeel, editors, *Proceedings of the 7th European Conference on Principles of Data Mining and Knowledge Discovery*, Lecture Notes in Artificial Intelligence. Springer-Verlag, 2003. To appear.

10. S. Orlando, P. Palmerini, R. Perego, and F. Silvestri. Adaptive and resource-aware mining of frequent sets. In V. Kumar, S. Tsumoto, P.S. Yu, and N.Zhong, editors, *Proceedings of the 2002 IEEE International Conference on Data Mining*, pages 338–345. IEEE Computer Society, 2002.
11. J.S. Park, M.-S. Chen, and P.S. Yu. An effective hash based algorithm for mining association rules. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, volume 24(2) of *SIGMOD Record*, pages 175–186. ACM Press, 1995.
12. D. Pavlov, H. Mannila, and P. Smyth. Beyond independence: probabilistic methods for query approximation on binary transaction data. *IEEE Transactions on Data and Knowledge Engineering*, To appear.
13. J. Pei, G. Dong, W. Zou, and J. Han. On computing condensed pattern bases. In *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM 2002), 9-12 December 2002, Maebashi City, Japan*, pages 378–385. IEEE Computer Society, 2002.
14. W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery. *Numerical Recipes in C++: The Art of Scientific Computing*. Cambridge University Press, 2002.
15. Z. Zheng, R. Kohavi, and L. Mason. Real world performance of association rule algorithms. In F. Provost and R. Srikant, editors, *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 401–406. ACM Press, 2001.

Queryable Lossless Log Database Compression

Kimmo Hätönen, Perttu Halonen, Mika Klemettinen, and Markus Miettinen

Nokia Research Center, P.O.Box 407, FIN-00045 Nokia Group, Finland
{kimmo.hatonen,perttu.halonen,mika.klemettinen,markus.miettinen}@nokia.com

Abstract This paper introduces a method for semantic compression of large log data collections. Unlike most of the well-known data compression methods, the proposed method maintain the readability of compressed log files. This makes it possible to query the compressed data set. The method is based on the use of *frequent closed sets*, which can be extracted from the data by several data mining algorithms. In this paper, we analyse the use of algorithm ACMINER for finding those sets. We show that it is computationally feasible to use even with large logs with hundreds of thousands of entries.

1 Introduction

Computer and communication networks and their monitoring applications are producing huge amounts of event logs. These event logs have to be stored for a certain period of time. Security related logs, for example, might be needed several months or even years after their creation in order to properly analyse security breaches that took place long time ago.

In practice this is difficult since the log files are huge: at worst millions of lines per day. Archiving of such data sets needs lots of machine resources. Even a bigger problem is to find specific data that are needed and load them from the archive to the current analysis environment.

The problem is not only with the size of logs but also with the number of them. In telecommunication systems, for example, there are tens of different log files and databases that should be available for different trouble shooting or security monitoring purposes. For example, event logs from Network Element operating systems, network alarm logs, and several security logs to name a few. New tools and new logs are introduced gradually.

Log files are typically archived in compressed form. Compression is, in many cases, done with the well-known Lempel-Ziv compression algorithm (LZ) [10] or with some other corresponding algorithm. When the log files are restored and a query or a regular expression search for relevant lines is made, the whole archive must be de-compressed.

Another possibility to archive history logs is to first insert them to a data base management system and then, after a certain period, compress the whole database table to a file that is inserted to a mass storage. The problem becomes real when there is a need to analyse old backups. An expert has to find the correct media, de-compress the whole database and load it to the data base management

system. This can be problematic because the amount of data included in a database per day might be large. Thus its de-compression and uploading takes a lot of time.

If the compression is not done in the archiving the problem is to fit the database to mass storage media and still be able to manage the rapidly grown number of mass storage media. Selecting carefully what is stored can reduce the problem, but still there tends to be quite a lot of data written into archives. Selection might also lose important information, which is not acceptable, for example, in security application logs.

In the first half of this paper we show how it is possible to use *closed sets* [3] to create a representation that is able to identify and code events and value combinations, which occur so frequently in the data that they contain most of the volume of the log. This coding reduces the size of the stored log. The coding can be done without any prior knowledge about the events. It doesn't lose any information and the original log file can easily be restored. This approach is more straightforward compared to the solutions proposed for semantic compression of databases [2] and documents [8]. Our approach is lossless and neither needs predefined knowledge about the data to run nor requires any kind of deduction to answer queries about the data.

In the latter half of this paper we discuss about possibilities to implement and optimize the proposed compression algorithm. Since the algorithms that find frequent closed sets possibly require exponential execution time with respect to the amount of analysed data, we have to be careful how they are used. We show how different factors affect to execution times of the selected algorithm called ACMINER [4,5].

In the following section we describe the proposed compression method. In Section 3 we present the theoretical background for using closed sets in the compression. In Section 4 we describe the experiments we did on algorithms ACMINER and APRIORI and show the experiment results. Finally, in Section 5, we present a conclusion and sketch further studies.

2 Queryable Lossless Log Compression

Comprehensive Log Compression (CLC) [7] is a method, which extracts metainformation from the log transactions and uses it to summarize redundant value combinations without losing any information. The method first analyses the log data and searches for *closed itemsets*. When linked with the transactions, which support them, these sets and their inclusion graph can be used as navigational links to the dataset.

Closed sets can further be used to create so called *condensation formulae*. These formulae are used to identify and remove repetitive value combinations from transactions. The condensation formulae are stored together with the remaining parts of the data. If the original table is needed, the formulae can be used to restore it completely – no information will be lost.

```
777;11May2000; 0:00:23;a_daemon;B1;12.12.123.12;tcp;;
778;11May2000; 0:00:31;a_daemon;B1;12.12.123.12;tcp;;
779;11May2000; 0:00:32;1234;B1;255.255.255.255;udp;;
780;11May2000; 0:00:38;1234;B2;255.255.255.255;udp;;
781;11May2000; 0:00:43;a_daemon;B1;12.12.123.12;tcp;;
782;11May2000; 0:00:51;a_daemon;B1;12.12.123.12;tcp;;
```

Figure1. An excerpt from a firewall log.

```
{Date:11May2000, Destination:255.255.255.255; proto:udp, service:1234, SPort:}  2
{Date:11May2000, Destination:12.12.123.12, Proto:tcp, Service:a_daemon, SPort:, Src:B1} 4
```

Figure2. Closed sets derived from the firewall log excerpt.

Figure 1, for example, shows an excerpt from a database containing transactions that store firewall log entries produced by CheckPoint's Firewall-1. As can be seen, the transactions are filled with entries that share correlating value combinations but still there are some fields whose values are varying; e.g., there are *TIME* and *ID* fields that are changing from line to line.

In Figure 2, there are closed sets that are needed in data compression. They contain those items that have the largest coverage, i.e., they cover the largest amount of field values in the table. These sets are called *compression patterns*. The method goes through each transaction. It compares the transaction to the compression patterns. If the transaction supports any of the patterns, values included in the pattern are removed from it and the rest of it is included to the archive together with a reference to the applied pattern. There are several strategies that can be used to construct the archive. In Figure 3 the compressed data are shown by using an XML-tagged format.

The method enables the optimisation of query mechanisms by reducing the number of lines to which a query or a regular expression has to be applied during the search for an answer. This is possible since the header that includes all the patterns, also includes the most frequently occurring values. While a query for the lines is evaluated, it is in many cases sufficient to query only the patterns and the lines that were not covered by any of them.

3 Closed Sets and Compression

We adopt here the definition of a *LOG pattern domain* as it is given by Hätönen et al. [7] It gives the definition of a language of patterns $\mathcal{L}$, evaluation functions that assign a description to each pattern in a given log $\mathbf{r}$, and languages for primitive constraints that specify the desired patterns. The definition uses notation of a so-called *log*, which contains the data in a form of *log entries* and patterns that are so-called *itemsets*, which are sets of $(field, value)$ pairs of log entries.

Here we enhance the definition of *LOG pattern domain* with concepts that are needed in log compression and de-compression phases.

```
<define p0> *;11May2000;*;a_daemon;B1;12.12.123.12;tcp;; </define>
<define p1> *;11May2000;*;1234;*;255.255.255.255;udp;; </define>
777; 0:00:23;<p0>
778; 0:00:31;<p0>
779; 0:00:32;B1;<p1>
780; 0:00:38;B2;<p1>
781; 0:00:43;<p0>
782; 0:00:51;<p0>
```

Figure3. Compressed firewall log excerpt

Definition 1 (Compression pattern). *A* compression pattern $\mathcal{C}_{\text{compr}}$ *is a* $\mathcal{C}_{close}(S, \mathbf{r})$ *such that* $\mathcal{C}ov(\mathcal{C}_{\text{compr}}) > \gamma$*, where* γ *is a given coverage threshold.*

Definition 2 (Compression formulae). *A* compression formulae $\mathcal{C}_{\text{formulae}}(\mathbf{r})$ *is a set of compression patterns* $\mathcal{C}_{\text{compr}_i}$ *derived from the log* $\mathbf{r}$*.*

Definition 3 (Compression cost). *The* compression cost *of a compression pattern* $\mathcal{C}_{\text{compr}}$ *in a log* $\mathbf{r}$ *is defined by* $\mathcal{CC}ost(\mathcal{C}_{\text{compr}}, \mathbf{r}) = (\mathcal{F}(\mathcal{C}_{\text{compr}}, \mathbf{r}) \cdot n) + |\mathcal{C}_{\text{compr}}|$*, where* n *denotes the size of the reference to the pattern, and* $|.|$ *denotes the cardinality of the compression pattern* $\mathcal{C}_{\text{compr}}$*.*

Definition 4 (Compression gain). *The* compression gain $\mathcal{CG}ain$ *of a compression pattern* $\mathcal{C}_{\text{compr}}$ *in a log* $\mathbf{r}$ *is defined by* $\mathcal{CG}ain(\mathcal{C}_{\text{compr}}, \mathbf{r}) = \mathcal{C}ov(\mathcal{C}_{\text{compr}}, \mathbf{r}) - \mathcal{CC}ost(\mathcal{C}_{\text{compr}}, \mathbf{r})$*.*

Input: Log L
Output: Compressed log $\mathcal{L}_{\text{compr}}$

1. Find frequent closed sets CFS
2. Collect compression formulae $\mathcal{C}_{\text{formulae}}(L)$
3. Filter log contents

Figure4. An algorithm for log compression.

As can be seen from Figure 4 the log compression is a straightforward operation. After the closed sets in the given log have been identified, the algorithm selects those, whose coverage is over a given threshold and whose compression gain is positive. This set of Compression Patterns is then used to remove recurrent value combinations away from the log entries. The removed values are replaced with a reference to the pattern that was used to identify them. The replacement is done w.r.t. the most specific, i.e., the longest, compression pattern that applies to the log entry.

The computational complexity of the algorithm mainly depends on the algorithm that searches for closed sets. Otherwise, the complexity is feasible. With

this in mind we have done series of experiments on factors that affect the execution of algorithms computing closed sets. The results of these experiments are reported in Section 4.

The de-compression of log data is as simple as the compression. The algorithm goes through the compressed log entries and expands them with the values of the compression pattern that was used for compression.

Input: Compressed Log $\mathcal{L}_{\text{compr}}$, Query $\mathcal{Q}$
Output: Requested set of log entries $\mathcal{RS}$.

1. **for** each pattern $p \in \mathcal{C}_{\text{formulae}}(L)$ **do**
2. **if** $\forall(f_i, value_i) \in \mathcal{Q} : \exists(f_i, value_j) \in p$ **then**
3. **if** $\forall i, j : value_i = value_j$ **then**
4. **for** each $l \in$ support(p) **do**
5. $\mathcal{RS} = \mathcal{RS} \cup \{l \setminus \{ref(p)\} \cup p\}$
6. **od**
7. **fi**
8. **elsif** $\forall f_i : (f_i, val_i) \in \mathcal{Q} \wedge (f_i, val_j) \in p$ applies that $val_i = val_j$ **then**
9. **if** matches$(l \cup p,\ \mathcal{Q})$, where $l \in$ support(p) **then**
10. $\mathcal{RS} = \mathcal{RS} \cup \{l \setminus \{ref(p)\} \cup p\}$
11. **fi**
12. **od**
13. **for** each entry $l \in \mathcal{L}_{\text{compr}}$ such that $\neg\exists p \in \mathcal{C}_{\text{formulae}}(L) \wedge ref(p) \in l$ **do**
14. **if** matches$(l, \mathcal{Q})$ **then**
15. $\mathcal{RS} = \mathcal{RS} \cup \{l\}$
16. **fi**
17. **od**
18. **return**$(\mathcal{RS})$.

Figure5. An algorithm for query evaluation in compressed log.

One of the main advantages of the proposed compression method is its ability to support queries in compressed form. The query evaluation is based on the use of compression patterns (see Figure 5). They capture not only the volume of the log data but also the value combinations. Therefore, it is possible to evaluate any query first against the compression patterns. If a pattern contains a value for all of the variables mentioned in the query and those values match for the queried values, then all the lines in the support of that pattern must be included in the answer. If the pattern contains only some of the queried variables and all those values match to the query, then all the lines in the support must be checked. Those entries that are not in a support of any of the compression patterns must be checked anyway.

The use of compression patterns in query evaluation decreases amount of entries that has to be checked. If a pattern contains such a value for a queried

variable that doesn't match to the query, then all the lines in its support can be left out from the answer. This reduces the search space quite efficiently.

There are several possible algorithms for finding closed sets [7]. We have adopted a solution that first finds so called *frequent free sets* and then produces their closures [4,5]. This is efficient since the freeness property is anti-monotonic, i.e., a key property for an efficient processing of the search space. Also, if compared to APRIORI-like algorithms [1], with this algorithm it is possible to minimize number of data base scans [9,3].

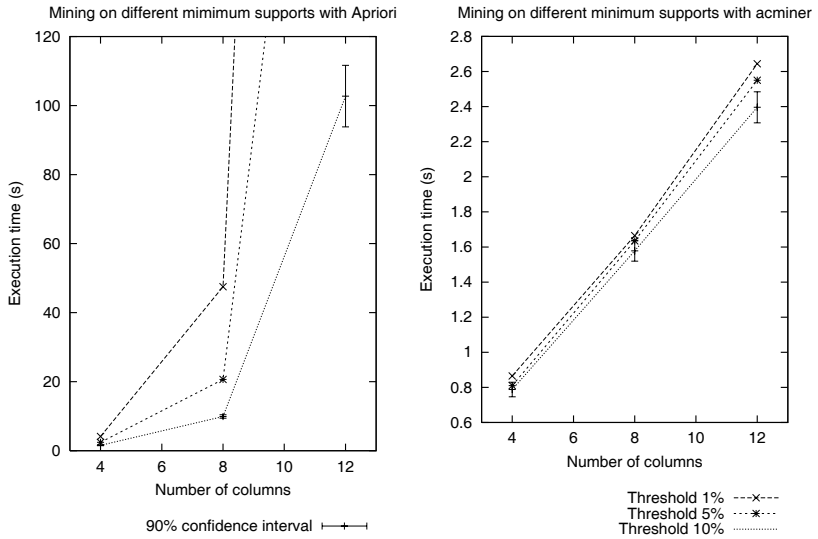


Figure6. Execution times of APRIORI (left) and ACMINER with the same input sets. Note the different scales on time axes.

In order to obtain a reference point, or a baseline, for the experiments we considered a combination of a well-known and well-studied algorithm APRIORI [1] and an algorithm for condensing the set of frequent sets. The results of these tests are shown in Figure 6. As was expected due to the well-known properties of the APRIORI algorithm, when the number of used database columns (input variables) was increased, the execution times of APRIORI increased by exponential fasion. This happens because the data contains lots of correlating value combinations and the number of candidate sets generated during each levelwise iteration grows rapidly. We used the same data to compute the closed sets with ACMINER [4,5]. As can be seen from Figure 6 the ACMINER is able to handle the growing amount of correlating values in the input data in linear fashion.

4 Experiments

In the experiments, we used firewall logs as input data. Firewall logs are a growing problem for operators. They are huge but in order to be able to track down possible incidents that have occurred already a while ago, they have to be stored for quite a long time.

A firewall log is a sparse transactional database described for example in [6]. Many columns in it potentially have a very large set of possible values, e.g., the originating address can be any IP address in the network. Simultaneously some of the columns may be very dense; they have only a few possible values from which one is much more common than the others. This means that the value distribution of a column follows a very skewed distribution. There are also lots of local correlations between values in columns. A high correlation together with the skewed exponential value distribution cause the number of frequent itemsets to increase dramatically.

As it was mentioned earlier, the algorithm that is used to find frequent closed sets is the main issue when the computational complexity of the proposed compression algorithm is studied. In our preliminary experiments, the search for frequent closed sets took more than half of the computation time in all cases. Therefore, we concentrate first on the ACMINER, which was selected as our closed sets algorithm. In order to evaluate the ACMINER algorithm, in our experiments we studied the effects of input size: the number of transactions and the number of columns in the input. We also tested the effects of the frequency threshold, since it is the most crucial parameter controlling the results of the algorithm.

Pure real-life data always have irregularities that make it difficult to compare the results with different sets of data. Therefore, it was necessary to modify the input data somewhat. We started the selection of input data by splitting the data mass to date files covering 24 hours each. The logging activity of a firewall depends much on the activity of people acting in the network, so the time of day matters and thus we handled the data in 24 hour units. Then, we selected logs of different sizes. We found three size categories in the logs we had at our disposal: 5050–5200, 13,000–14,000 and 42,000–62,000 transactions. Last, we studied their statistical properties and made modified versions of them by dropping out some of the columns in order to embody the factor number of columns. When we altered the number of columns, we first removed the transaction number, date and time fields as they are quite uniformly distributed and thus uninteresting. Then we randomly chose the others we dropped out, thus ensuring that the distributions on average didn't change.

We did the experiments with sixteen factor combinations. We combined different minimum support thresholds and amounts of columns and kept the number of transactions between 13,000–14,000, and different amounts of columns and transactions and kept the minimum support threshold constant. We made ten files for each factor combination in order to reduce the influence of a particular input to the measurement results. After these tests we analysed separately a set of ten very large logs containing 20 columns and more than 500,000 transactions each.

We ran the tests in a Compaq Deskpro workstation with a 500 MHz Pentium III chip, 380 MB of memory, 2 GB of swap space, and running Linux. We didn't use it in the single user mode, but minimised all the other load on it during the experiments.

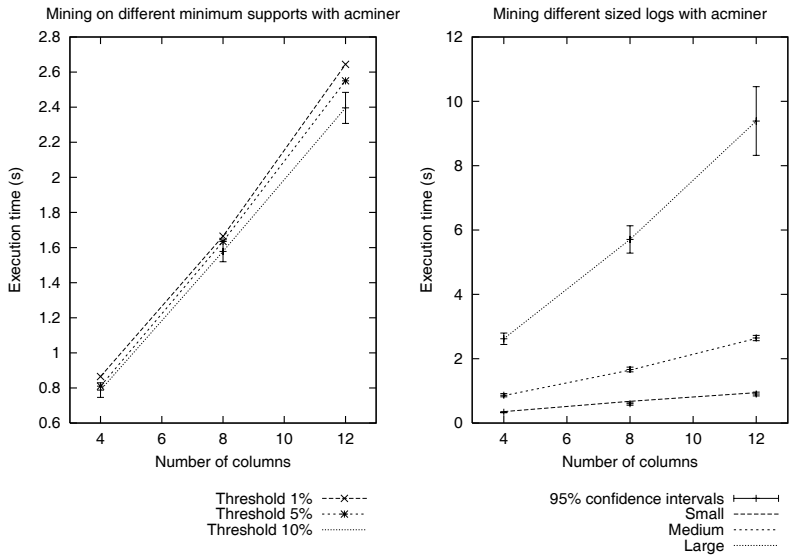


Figure7. Execution times of ACMINER with medium sized input on different minimum support thresholds (left) and with minimum support threshold of 1.0% on different input sizes.

In first nine test cases we tested the impact of different minimum support thresholds and numbers of columns. We used three threshold values: $1.0\%, 5.0\%$ and 10.0% of the number of transactions, and the number of columns was $4, 8$ and 12. The size category was the medium class (13,000–14,000 transactions).

When the number of columns in data was increased the execution times of ACMINER grew in linear fashion, as can be seen from Figure 7. In the graph on the left we have plotted the means of the execution times of the algorithm. As can be seen from confidence limits the results within a test set deviate so much that the impact of increasing the minimum support threshold is not clear. This looks quite promising with respect to the ability to gain more powerful compression with lower frequency thresholds.

In the next six test cases we studied the influence of the input size. We used small and large sized input with $4, 8$ and 12 columns. The minimum support threshold was constantly 1.0%. In the graph on the right in Figure 7, we have plotted the execution times of the algorithm on different input sizes. The number

of columns seems to have very clear impact on the execution time as the quite reliable 95% confidence intervals do not cross each other. The size of the input seems to influence the execution time by a constant value that is influenced by the number of columns.

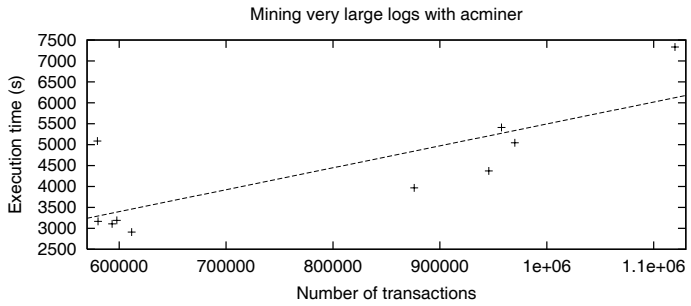


Figure8. Execution times of ACMINER on very large inputs.

In Figure 8, the execution times of the algorithm are plotted as a function of the number of transactions. We have used the very large logs as input. We have also drawn a regression line that suggests that the size of input really has a linear influence to the execution time. The minimum support threshold was 1.0%.

Beside the computational feasibility of the proposed algorithm another interesting feature of it is the achieved compression rate. To estimate this we used the medium size test data set as it was described above. The results were encouraging. The average compression ratio was 69%, when the lowest ratio that was achieved was 58% and highest 83%. The most interesting observation made during the testing was that the compression ratio that was achieved by our implementation is not optimal. This is because it is not able to handle compression patterns that are partially overlapping. The implementation selects the one that is the most specific, and makes the compression of an entry with it. The optimal solution would be to use the union of all applicable compression patterns as the compression formula. We will continue to study this in the near future.

5 Conclusion

In this paper, we proposed the method for queryable lossless compression of log databases. It uses ACMINER as a tool to find closed sets, which it uses as a basis for compression mechanism. We tested the algorithm with real-life firewall data. Our goal was to ensure the applicability and scalability of the proposed Log Compression method (see Section 2) with different types of data sets. ACMINER algorithm appears to be feasible for our purposes when the execution time is

concerned, as can be seen from the results in Section 4. In the near future we will continue to study and develop the proposed method and experiment with other real-life data sets.

Acknowledgements

This study is a part of Nokia Research Center's activity in the *consortium on discovering knowledge with Inductive Queries* (cInQ) and will be a basis for further research and articles. The cInQ project is funded by the Future and Emerging Technologies arm of the IST Programme (Contact no. IST-2000-26469).

We want to thank professor Jean-François Boulicaut, Dr. Artur Bykowski, and Institut National des Sciences Appliquées de Lyon for the implementation of the acminer algorithm.

References

1. Rakesh Agrawal, Heikki Mannila, Ramakrishnan Srikant, Hannu Toivonen, and A. Inkeri Verkamo. Fast discovery of association rules. In Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, and Ramasamy Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 307 – 328. AAAI Press, Menlo Park, CA, 1996.
2. Shivnath Babu, Minos Garofalakis, and Rajeev Rastogi. Spartan: A model-based semantic compression system for massive data tables. In *Proceedings of ACM-SIGMOD 2001*, Santa Barbara, California, May 2001. ACM.
3. Jean-François Boulicaut and Artur Bykowski. Frequent closures as a concise representation for binary data mining. In *Proceedings PAKDD'00*, volume 1805 of *LNAI*, pages 62–73, Kyoto, JP, April 2000. Springer-Verlag.
4. Jean-François Boulicaut, Artur Bykowski, and Christophe Rigotti. Approximation of frequency queries by mean of free-sets. In *Proceedings PKDD'00*, volume 1910 of *LNAI*, pages 75–85, Lyon, F, September 2000. Springer-Verlag.
5. Jean-François Boulicaut, Artur Bykowski, and Christophe Rigotti. Free-sets: a condensed representation of boolean data for the approximation of frequency queries. *Data Mining and Knowledge Discovery journal*, 7(1):5–22, 2003.
6. Jean-François Boulicaut and Baptiste Jeudy. Mining free-sets under constraints. In *Proceedings IDEAS'01*, pages 322–329, Grenoble, F, July 2001. IEEE Computer Society.
7. Kimmo Hätönen, Jean François Boulicaut, Mika Klemettinen, Markus Miettinen, and Cyrille Masson. Comprehensive log compression with frequent patterns. In *Proceedings of Data Warehousing and Knowledge Discovery - DaWaK 2003 (DaWaK'03)*, Prague, Czech Republic, Sept 2003. Springer-Verlag.
8. Hartmut Liefke and Dan Suciu. XMill: an efficient compressor for XML data. In *Proceedings of ACM-SIGMOD 2000*, pages 153–164, Dallas, Texas, June 2000. ACM.
9. Nicolas Pasquier, Yves Bastide, Rafik Taouil, and Lotfi Lakhal. Efficient mining of association rules using closed itemset lattices. *Information Systems*, 24(1):25–46, January 1999.
10. J. Ziv and A. Lempel. A universal algorithm for sequental data compression. *IEEE Transactions on Information Theory*, 22(1):337–343, 1977.

An Algebra for Inductive Query Evaluation

Sau Dan Lee and Luc De Raedt

Institut für Informatik
Albert-Ludwigs-Universität Freiburg
Georges-Köhler-Allee, Gebäude 079
D-79110 Freiburg
Germany
{danlee,deraedt}@informatik.uni-freiburg.de

Abstract. Inductive queries are queries that generate pattern sets. This paper studies properties of boolean inductive queries, i.e. queries that are boolean expressions over monotonic and anti-monotonic constraints. More specifically, we introduce and study algebraic operations on the answer sets of such queries and show how these can be used for constructing and optimizing query plans. Special attention is devoted to the dimension of the queries, i.e. the minimum number of version spaces (or convex sets) needed to represent the answer sets. The theoretical framework has been implemented for the pattern domain of strings.

1 Introduction

Many data mining problems address the problem of finding a set of patterns that satisfy a constraint. Formally, this can be described as the task of finding the set of patterns $Th(Q, \mathcal{D}, \mathcal{L}) = \{\varphi \in \mathcal{L} \mid Q(\varphi, \mathcal{D})\}$, i.e. those patterns φ satisfying query Q on database $\mathcal{D}$. Here $\mathcal{L}$ is the language in which the patterns or rules are described and Q is a predicate or constraint that determines whether a pattern φ is a solution to the data mining task or not [1]. This framework allows us to view the predicate or the constraint Q as an *inductive query* to an *inductive database system* [2]. It is then the task of the inductive database management system to efficiently generate the answers to the query. This view of data mining as a declarative querying process is also appealing as the basis for a theory of data mining. Such a theory would be analogous to traditional database theory in the sense that one could study properties of different pattern languages $\mathcal{L}$, different types of queries (and query languages), as well as different types of databases. Such a theory could also serve as a sound basis for developing algorithms that solve and optimize inductive queries.

When developing a theory about inductive databases, the relational algebra may provide a useful source of inspiration. The relational algebra has proven to be useful for database theory because of a variety of reasons. First, there is the so-called closure property, which states that the result of a query is a relation. This allows the result of one query to be used in the next query. Secondly, relational algebra allows one to reason about query execution and optimization. Indeed,

before executing a query, the database management system constructs and possibly optimizes a so-called query plan. Thirdly, the relational algebra relies on a simple yet powerful set of mathematical primitives. Because inductive databases aim at putting data mining on the same methodological grounds as databases, it should be useful to develop an algebra for manipulating pattern sets. Pattern sets are the solutions to inductive queries $Th(Q, \mathcal{D}, \mathcal{L})$. Following the analogy outlined above, they are the equivalent of a relation in a relational database. This paper contributes an algebraic framework for manipulating pattern sets and inductive queries. It extends the theoretical framework for inductive querying by De Raedt *et al.* [3], who have contributed a theory centered around the class of boolean inductive queries. Boolean inductive queries are queries that are boolean expressions over monotonic and anti-monotonic predicates. This class is—to the best of the authors' knowledge—the most expressive class of queries considered so far. The present paper is an extension as we consider algebraic set operations on pattern sets and study their properties. The operations that we study include intersection, union, difference and complements. They directly correspond to conjunction, disjunction, negated implication and negation. A central concept in this theory is the notion of version space [4]. A version space is a convex set, which can be represented by its border sets (of maximally general and maximally specific elements). The solution space corresponding to a conjunctive query is a version space. Furthermore, effective algorithms exist for computing these solution sets [5, 6, 3, 7, 8]. In this context, De Raedt *et al.* have studied the dimension of pattern sets (and boolean queries), i.e. the minimum number of version spaces needed to represent the pattern set. This in turn is related to the number of calls to a conjunctive query answering algorithm needed to answer an inductive query. In the present paper, these results are extended in the light of the algebraic operations and we also show how these results can be employed when reasoning about query execution and optimization.

Version spaces have been introduced in the context of concept-learning by Tom Mitchell [4] and have been rather popular ever since. Hirsh [9] has investigated various set operations (such as intersection and union) on version spaces and employed these in the context of concept-learning and explanation based learning [10]. On the theoretical side, he has shown that version spaces are closed under intersection but not under union. From this perspective, our present results extend those by Hirsh in that we show that generalized version spaces (i.e. finite unions of version spaces) are closed under set operations and that we also deduce bounds on the dimensions of the resulting sets.

Finally, as an illustration of our general framework, we apply it to the pattern domain of strings. For this domain, we have designed a data structure, called the generalized version space tree, that is useful for computing and memorizing the solution space w.r.t. inductive queries. The pattern domain of strings is of interest because of the rapid generation of biological string databases, e.g. about DNA, genes or proteins.

This paper is organized as follows. Sect. 2 introduces the problem of boolean inductive queries, closely following [3]. Sect. 3 then studies the algebraic opera-

tions, which enables us to construct many equivalent query execution plans for a given query (Sect. 4) and seek the optimal one. We implemented our framework on the domain of string patterns with the help of our novel data structure, the generalized version space tree (Sect. 5), and performed some preliminary experiments on two data sets (Sect. 6). Finally, we conclude in Sect. 7.

2 Boolean Inductive Queries

Let us first define boolean inductive queries. We closely follow De Raedt et al. [3].

A pattern language $\mathcal{L}$ is a formal language for specifying patterns. Each pattern $\varphi \in \mathcal{L}$ matches (or covers) a set of examples φ_e, which is a subset of the universe $\mathcal{U}$ of possible examples.

Example 1. Let Σ be a finite alphabet and $\mathcal{U}_\Sigma = \Sigma^*$ the universe of all strings over Σ. We will denote the empty string with ϵ. The traditional pattern language in this domain is $\mathcal{L}_\Sigma = \mathcal{U}_\Sigma$. A pattern $\varphi \in \mathcal{L}_\Sigma$ covers the set $\varphi_e = \{\psi \in \Sigma^* \mid \varphi \sqsubseteq \psi\}$, where $\varphi \sqsubseteq \psi$ denotes that φ is a substring of ψ. An alternative, more expressive, language is the language of all regular expressions over Σ.

One pattern φ is *more general* than a pattern ψ, written $\varphi \preceq \psi$, if and only if $\varphi_e \supseteq \psi_e$.

A pattern *predicate* defines a primitive property of a pattern, usually relative to some data set $\mathcal{D}$ (a set of examples), and sometimes other parameters. For any given pattern, it evaluates to either *true* or *false*.

We now introduce a number of pattern predicates that will be used for illustrative purposes throughout this paper. Most of these predicates are inspired by MolFea [6]. Our first pattern predicates are very general in that they can be used for arbitrary pattern languages:

- $\mathsf{min\_freq}(\varphi, D, n)$ evaluates to true iff φ is a pattern that occurs in database $\mathcal{D}$ with frequency at least n. The frequency $\mathit{freq}(\varphi, \mathcal{D})$ of a pattern φ in a database $\mathcal{D}$ is the (absolute) number of data items in $\mathcal{D}$ covered by φ. Analogously, the predicate $\mathsf{max\_freq}(\varphi, \mathcal{D}, n)$ is defined.
- $\mathsf{is\_more\_general}(\varphi, \psi)$ is a predicate that evaluates to true iff pattern $\varphi \preceq \psi$. Dual to the is_more_general predicate one defines the is_more_specific predicate.

The following predicate is an example predicate tailored towards the specific domain of string-patterns over $\mathcal{L}_\Sigma$.

- $\mathsf{length\_at\_most}(\varphi, n)$ evaluates to true for $\varphi \in \mathcal{L}_\Sigma$ iff φ has length at most n. Analogously the $\mathsf{length\_at\_least}(\varphi, n)$ predicate is defined.

In all the preceding examples the predicates on the patterns have the form $\mathsf{predicate}(\varphi, \mathit{params})$ or $\mathsf{predicate}(\varphi, \mathcal{D}, \mathit{params})$, where *params* is a tuple of parameter values, $\mathcal{D}$ is a data set and φ is a pattern variable. We also speak a bit

loosely of predicatealone as a pattern predicate, and mean by that the collection of all pattern predicates obtained for different parameter values *params*. We say that m is a *monotonic* predicate, if for all possible parameter values *params* and all data sets D:

$$\forall \varphi, \psi \in \mathcal{L} \text{ such that } \varphi \preceq \psi : \mathsf{m}(\varphi, \mathcal{D}, params) \rightarrow \mathsf{m}(\psi, \mathcal{D}, params)$$

The class of *anti-monotonic* predicates is defined dually. Thus, min_freq, is_more_general, and length_at_most are monotonic, their duals are anti-monotonic.

A pattern predicate $\mathsf{predicate}(\varphi, \mathcal{D}, params)$ that can be applied to the patterns from a language $\mathcal{L}$ defines relative to $\mathcal{D}$ the *solution set* $Th(\mathsf{predicate}(\varphi, \mathcal{D}, params), \mathcal{L}) = \{\phi \in \mathcal{L} \mid \mathsf{predicate}(\varphi, \mathcal{D}, params) = true\}$. Furthermore, for monotonic predicates m these sets will be monotone, i.e. for all $\phi \succeq \psi \in \mathcal{L} : \psi \in Th(\mathsf{m}, \mathcal{L}) \rightarrow \phi \in Th(\mathsf{m}, \mathcal{L})$.

Example 2. Consider the string data set $\mathcal{D}$= {abc, abd, cd, d, cd}. Here we have pattern frequencies $freq(\mathtt{abc}, D) = 1$, $freq(\mathtt{cd}, D) = 2$, $freq(\mathtt{c}, D) = 3$, $freq(\mathtt{abcd}, D) = 0$. And trivially, $freq(\epsilon, D) = |D| = 5$. Thus, the following predicates evaluate to true: min_freq(c, $\mathcal{D}$, 2), min_freq(cd, $\mathcal{D}$, 2), max_freq(abc, $\mathcal{D}$, 2), max_freq(cd, $\mathcal{D}$, 2).

The pattern predicate $\mathsf{m} := \mathsf{min\_freq}(\varphi, \mathcal{D}, 2)$ defines $Th(\mathsf{m}, \mathcal{L}_\Sigma) = \{\epsilon, a, b, c, d, ab, cd\}$, and the pattern predicate $\mathsf{a} := \mathsf{max\_freq}(\varphi, \mathcal{D}, 2)$ defines the infinite set $Th(\mathsf{a}, \mathcal{L}_\Sigma) = \mathcal{L}_\Sigma \setminus \{\epsilon, c, d\}$.

Definition 3. *The definition of* $Th(\mathsf{predicate}(\varphi, \mathcal{D}, \text{params}), \mathcal{L})$ *is extended in the natural way to a definition of the solution set* $Th(Q, \mathcal{L})$ *for boolean combinations* Q *of pattern predicates over a unique pattern variable:* $Th(\neg Q, \mathcal{L}) := \mathcal{L} \setminus Th(Q, \mathcal{L})$*;* $Th(Q_1 \vee Q_2, \mathcal{L}) := Th(Q_1, \mathcal{L}) \cup Th(Q_2, \mathcal{L})$*;* $Th(Q_1 \wedge Q_2, \mathcal{L}) := Th(Q_1, \mathcal{L}) \cap Th(Q_2, \mathcal{L})$*. The predicates that appear in* Q *may reference one or more data sets* $\mathcal{D}_1, \ldots, \mathcal{D}_n$*.*

We are interested in computing solution sets $Th(Q, \mathcal{D}, \mathcal{L})$ for boolean queries Q that are constructed from monotonic and anti-monotonic pattern predicates. We can thus formally define the *boolean inductive query evaluation problem* addressed in this paper.

Given
- a language $\mathcal{L}$ of patterns,
- a set of monotonic predicates $\mathcal{M} = \{\mathsf{m}_1(p, D_1, params_1), \ldots, \mathsf{m}_n(p, D_j, params_j)\}$ and a set of anti-monotonic predicates $\mathcal{A} = \{\mathsf{a}_1(p, D_1, params_1), \ldots, \mathsf{a}_k(p, D_k, params_k)\}$
- a query Q that is a boolean expression over the predicates in $\mathcal{M}$ and $\mathcal{A}$.

Find the set of patterns $Th(Q, \mathcal{D}, \mathcal{L})$, i.e. the solution set of the query Q in the language $\mathcal{L}$ with respect to the data subsets $D_1, \ldots, D_n \subseteq \mathcal{D}$.

De Raedt *et al.* have proposed a strategy for evaluating inductive queries and also a first step in the direction of query optimization. Their strategy consists of

decomposing a boolean query Q into k sub-queries Q_i such that Q is equivalent to $Q_1 \vee \ldots \vee Q_k$, k is minimal and each of the Q_i is the conjunction of a monotonic and an anti-monotonic subquery $Q_{m,i} \wedge Q_{a,i}$. Notice that $Q_{m,i}$ and $Q_{a,i}$ may be boolean expressions themselves. Indeed, [11] show that $Q_{m,i}$ may be a DNF formula in which all the literals are monotonic, and similarly for $Q_{a,i}$. A DNF formula of the form $M_1 \vee \ldots \vee M_n$ where each of the M_i is a conjunction of monotonic predicates is monotonic. The reason is that the conjunction of monotonic predicates is monotonic (hence all of the M_i are), and also the disjunction of monotonic predicates is monotonic as well (and hence $Q_{m,i}$ is). The query evaluation strategy proposed by [3] first decomposes the query and then computes $Th(Q, \mathcal{D}, \mathcal{L})$ as $\cup_i Th(Q_i, \mathcal{D}, \mathcal{L})$. Because each of the sub-queries Q_i will be such that $Th(Q_i, \mathcal{D}, \mathcal{L})$ is a version space (also called a convex space), it can be efficiently computed for a wide class of pattern languages $\mathcal{L}$, and queries Q_i, cf. [6, 7, 12]. Furthermore, the number of calls to such an algorithm is minimized because in the decomposition process, one minimizes the number k of subqueries. This is the so-called dimension of a query, which [3] introduced as follows:

Definition 4. *Let $\mathcal{L}$ be a pattern language, and $X \subseteq \mathcal{L}$. Then X is a version space of* dimension *1, if $\forall \varphi, \varphi', \psi \in \mathcal{L} : \varphi \preceq \psi \preceq \varphi'$ and $\varphi, \varphi' \in X \implies \psi \in X$. The set of all version spaces of dimension 1 for $\mathcal{L}$ is denoted $\mathcal{VS}^1(\mathcal{L})$.*

Definition 5. *The subset of patterns in $\mathcal{L}$ that satisfy the predicate Q, composed of monotonic or anti-monotonic predicates using a combination of the 3 logical operators conjunction ($\wedge$), negation ($\neg$) and disjunction ($\vee$), is a* generalized version space*, denoted by $Th(Q, \mathcal{D}, \mathcal{L})$. The set of all generalized version spaces for $\mathcal{L}$ is denoted by $\mathcal{VS}^{\mathbb{Z}}(\mathcal{L})$. Note that $\mathcal{VS}^1(\mathcal{L}) \subseteq \mathcal{VS}^{\mathbb{Z}}(\mathcal{L})$.*

Definition 6. *The set $X \in \mathcal{VS}^{\mathbb{Z}}(\mathcal{L})$ has dimension k if it is the union of k sets in $\mathcal{VS}^1$, but not the union of $k-1$ sets in $\mathcal{VS}^1$.*

A query Q has dimension k (with respect to the pattern language $\mathcal{L}$) if k is the maximal dimension of any solution set $Th(Q, \mathcal{D}, \mathcal{L})$ of Q (where the maximum is taken w.r.t. all possible data sets $\mathcal{D}$ and w.r.t. the fixed language $\mathcal{L}$).

If Q has dimension 1 w.r.t. $\mathcal{L}$, then $Th(Q, \mathcal{D}, \mathcal{L})$ is a version space [4] or a convex space [10]. Version spaces are particularly useful when they can be represented by boundary sets, i.e. by the sets $G(Q, \mathcal{D}, \mathcal{L})$ of their maximally general elements, and $S(Q, \mathcal{D}, \mathcal{L})$ of their minimally general elements. For the theoretical framework of the present section we need not assume boundary representability for convex sets. However, concrete instantiations of the general method we here develop, like the one described in sections 5 and 6, usually will assume pattern languages in which convexity implies boundary representability.

Example 7. Reconsider the string domain. Let

$$\begin{aligned} Q_1 &= \mathsf{is\_more\_general}(\varphi, \mathtt{abcde}) \wedge \mathsf{length\_at\_least}(\varphi, 3) \\ Q_2 &= \mathsf{is\_more\_specific}(\varphi, \mathtt{ab}) \wedge \mathsf{is\_more\_specific}(\varphi, \mathtt{uw}) \\ &\quad \wedge (\mathsf{length\_at\_least}(\varphi, 6) \vee \mathsf{min\_freq}(\varphi, \mathcal{D}, 3)) \end{aligned}$$

The query Q_1 does not reference any dataset, and $Th(Q_1, \mathcal{L}_\Sigma) = \{$ `abcde`, `abcd`, `bcde`, `abc`, `bcd`, `cde` $\}$. This set of solutions is completely characterized by $S(Q_1, \mathcal{L}_\Sigma) = \{\texttt{abcde}\}$ and $G(Q_1, \mathcal{L}_\Sigma) = \{$ `abc`, `bcd`, `cde` $\}$. $Th(Q_2, D, \mathcal{L}_\Sigma)$ cannot in general be represented using a single version space. However, as our general method will show, the dimension of $Th(Q_2, D, \mathcal{L}_\Sigma)$ is at most two, so that it can be represented as the union of two version spaces.

With the following definition and lemma, De Raedt *et al.* [3] provided an alternative characterization of dimension k sets.

Lemma 8. *Let $X \subseteq \mathcal{L}$. Call a chain $\varphi_1 \preceq \rho_1 \preceq \varphi_2 \preceq \rho_2 \preceq \cdots \preceq \varphi_{k-1} \preceq \rho_{k-1} \preceq \varphi_k$ an* alternating chain (of length k) for X *if $\varphi_i \in X$ $(i = 1, \ldots, k)$ and $\rho_j \notin X$ $(j = 1, \ldots, k-1)$. Then the dimension of X is equal to the maximal k for which there exists in $\mathcal{L}$ an alternating chain of length k (or a k-chain for short) for X.*

Example 9. Consider the following queries:

$$Q_3 = \mathsf{is\_more\_general}(\varphi, \texttt{abc}) \wedge \mathsf{is\_more\_specific}(\varphi, \texttt{a})$$
$$Q_4 = \mathsf{is\_more\_general}(\varphi, \texttt{c})$$
$$Q_5 = Q_3 \vee Q_4$$

Then `c`, `bc`, `abc` is an alternating chain of length 2 for $Th(Q_5, \mathcal{L}_\Sigma)$.

The dimension of an inductive query is an important concept because it corresponds to the minimum number of calls one has to make to an algorithm for computing a convex solution space.

3 Operations on Solution Spaces

Consider two generalized version spaces $V, W \in \mathcal{VS}^{\mathbb{Z}}$, We would like to know how we can combine them using the usual set operations. This is of interest because the solution set to an inductive query is, by Definition 3, obtained as the result of applying set operations on solution set of simpler queries. At the same time, these operations promise to be useful when constructing query plans and in interactive querying sessions (cf. Sect. 4 for a detailed discussion).

We will analyze the operations: intersection ($\cap$), union ($\cup$), and complement ($'$) w.r.t. $\mathcal{L}$. Set difference can be treated as $V \setminus W \equiv V \cap W'$ whereas symmetric difference can be interpreted as $(V \setminus W) \cup (W \setminus V)$.

When $\dim(V) = \dim(W)$ are both 1, we reduce to traditional version spaces $\mathcal{VS}^1$. We know from the previous discussion that $V \cap W$ is in $\mathcal{VS}^1$. We have also shown above by a counter-example that the dimension of the union can be 2, although it may also be 1 (e.g. when $V \subseteq W$). So, traditional version spaces, i.e. $\mathcal{VS}^1$, are not closed under union (as shown by [9]).

Nevertheless, our extension $\mathcal{VS}^{\mathbb{Z}}$ is closed under the usual set operations. The following theorems thus generalize Hirsh's results to the case where solution sets are represented by a finite number of version spaces.

Theorem 10. *$\mathcal{VS}^{\mathbb{Z}}(\mathcal{L})$ is closed under the following set operations: intersection, union, complement, difference and symmetric difference.*

It suffices to analyze the first 3 of them, as the latter can be defined out of the first 3 operations. The proofs will be given below together with bounds for resulting dimensions.

Theorem 11. *Let $V, W \in \mathcal{VS}^{\mathbb{Z}}(\mathcal{L})$. Then, $U = V \cap W \in \mathcal{VS}^{\mathbb{Z}}(\mathcal{L})$. Moreover, $0 \leq \dim(U) \leq \dim(V) + \dim(W) - 1$.*

Proof. Let $v = \dim(V)$ and $w = \dim(W)$. By Definition 6, we can write $V = \bigcup_{i=1}^{v} X_i$ and $W = \bigcup_{k=1}^{w} Y_k$ for some $X_i, Y_k \in \mathcal{VS}^1(\mathcal{L})$. Then, we have $U = (\bigcup_{i=1}^{v} X_i) \bigcap (\bigcup_{k=1}^{w} Y_k) = \bigcup_{\substack{i=1,\dots v \\ k=1,\dots w}} Z_{i,k}$ where $Z_{i,k} = X_i \cap Y_k$. Since every $Z_{i,k} \in \mathcal{VS}^1(\mathcal{L})$, we have $U \in \mathcal{VS}^{\mathbb{Z}}(\mathcal{L})$.

For the bounds on the dimension, in case V and W are disjoint, we will have an empty U, giving $\dim(U) = 0$, which is a lower bound. For the upper bound, we prove by contradiction. Assume that $U = V \cup W$ has dimension at least $u = v + w$. Then, by Lemma 8, U has a u-chain (or longer), i.e.

$$\exists \varphi_i \in U (i = 1, \dots, u)\ \exists \rho_j \in \mathcal{L} \setminus U (j = 1, \dots, u-1)$$
$$\varphi_1 \preceq \rho_1 \preceq \varphi_2 \preceq \rho_2 \preceq \cdots \preceq \rho_{u-1} \preceq \varphi_u$$

Denote the sequence $\{\rho_j\}_{j=1}^{u-1}$ by R. How many members of R are not a member of V? At most $v - 1$: if not, then there will be (at least) v members of R not belonging to V. Let these be ρ_{j_k} $(k = 1, \dots, v)$, where $j_k \in \{1, \dots, u-1\}$ and $\{j_k\}$ is an increasing sequence. Then, $\varphi_{j_1} \preceq \rho_{j_1} \preceq \varphi_{j_2} \preceq \rho_{j_2} \preceq \ldots \preceq \varphi_{j_v} \preceq \rho_{j_v} \preceq \varphi_{j_v+1}$ will be a $(v+1)$-chain for V, and hence by Lemma 8, $\dim(V) \geq v + 1$, which contradicts with our definition of v. Therefore at most $v - 1$ members of R do not belong to V. This means that at least $(u-1) - (v-1) = w$ members of R must belong to V. They form a subsequence $R_V = \{\rho_{p_r}\}_{r=1}^{p}$ of R, where $\{p_r\}_{r=1}^{p}$ is an increasing sequence on $\{0, \dots, u-1\}$. R_V is thus a sequence of length $p \geq w$.

By a similar argument, at least v members of R must belong to W. They form a subsequence $R_W = \{\rho_{q_s}\}_{s=1}^{q}$ of R ($\{q_s\}_{s=1}^{q}$ is an increasing sequence on $\{0, \dots, u-1\}$), with length of $q \geq v$. Now, the sequences R_V and R_W together has at least $v + w = u$ members, all coming from R. But R has only $u - 1$ members. By the pigeon hole principle, at least one member of R is common to both R_W and R_V. With our notations, this means there are r' and s' in $\{1, \dots, u-1\}$ such that $p_{r'} = q_{s'}$. Let $\rho' = \rho_{p_{r'}} = \rho_{q_{s'}}$. This is then a common member of R_V and R_W. According to the definition of R_W and R_V, $\rho' \in V \cap W = U$. But this contradicts with the fact that $\rho_j \in \mathcal{L} \setminus U$ for all $j = 1, \dots, u-1$. □

Theorem 12. *Let $V, W \in \mathcal{VS}^{\mathbb{Z}}(\mathcal{L})$. Then, $U = V \cup W \in \mathcal{VS}^{\mathbb{Z}}(\mathcal{L})$. Moreover, $0 \leq \dim(U) \leq \dim(V) + \dim(W)$.*

Proof. Let $v = \dim(V)$ and $w = \dim(W)$. By Definition 6, we can write $V = \bigcup_{i=1}^{v} X_i$ and $W = \bigcup_{k=1}^{w} Y_k$ for some $X_i, Y_k \in \mathcal{VS}^1(\mathcal{L})$ Then, we have $U =$

$(\bigcup_{i=1}^{v} X_i) \cup (\bigcup_{k=1}^{w} Y_k) = \bigcup_{l=1}^{v+w} Z_k$ where $Z_l = X_l$ for $l = 1, \ldots, v$ and $Z_{l-v} = Y_{l-v}$ for $l = v+1, \ldots, v+w$. Since each $Z_k \in \mathcal{VS}^1(\mathcal{L})$, we have $U \in \mathcal{VS}^{\mathbb{Z}}(\mathcal{L})$. Moreover, $\dim(U) \leq w+v$ by Definition 6. We cannot give a tighter lower bound other than 0, because of the following special cases. First, consider the case where $V \cup W = \mathcal{L}$, e.g. when $V = W'$. Then, we have $U = \mathcal{L} = Th(true, \mathcal{D}, \mathcal{L})$. Since "true" is a monotonic[1], $\dim(U) = 1$. The other special case is when both V and W are empty. This gives an empty U, and hence $\dim(U) = 0$. Therefore, $0 \leq \max(\dim(U), \dim(V)) \leq \dim(U) \leq \dim(V) + \dim(W)$. □

Lemma 13. *Let $V \in \mathcal{VS}^{\mathbb{Z}}(\mathcal{L})$. Then, $U = \mathcal{L} \setminus V \in \mathcal{VS}^{\mathbb{Z}}(\mathcal{L})$ and $\dim(U) \geq \dim(V) - 1$*

Proof. Let $v = \dim(V)$. By Lemma 8, V has a v-chain. i.e.

$$\exists \varphi_i \in V (i = 1, \ldots, v)\ \exists \rho_j \in \mathcal{L} \setminus V (j = 1, \ldots, v-1)$$
$$\varphi_1 \preceq \rho_1 \preceq \varphi_2 \preceq \rho_2 \preceq \cdots \preceq \varphi_{v-1} \preceq \rho_{v-1} \preceq \varphi_v$$

Then, $\rho_1 \preceq \varphi_2 \preceq \rho_2 \preceq \cdots \preceq \varphi_{v-1} \preceq \rho_{v-1}$ constitutes a $(v-1)$-chain of $U = \mathcal{L} \setminus V$. Thus, with Lemma 8, we conclude that $\dim(U) \geq v - 1$. □

Theorem 14. *Let $V \in \mathcal{VS}^{\mathbb{Z}}(\mathcal{L})$ and $v = \dim(V)$. Then, $U = \mathcal{L} \setminus V \in \mathcal{VS}^{\mathbb{Z}}(\mathcal{L})$ and $u = \dim(U) = v - 1, v$ or $v + 1$.*

Proof. There is a special case: $V = \emptyset$. In this case $\dim(V) = 0$, and $\mathcal{L} \setminus V = \mathcal{L}$, which has dimension 1. So, the theorem holds.

For the general case, Lemma 13 tells us that $u \geq v - 1$, i.e. $v - 1 \leq u$. Since $V = \mathcal{L} \setminus U$, we can apply the same lemma to obtain $\dim(V) \geq \dim(U) - 1$, i.e. $v \geq u - 1$, or equivalently, $u \leq v + 1$ Combining these results gives $v - 1 \leq u \leq v + 1$. Since u must be integer, it must take the value of one of $v - 1$, v or $v + 1$. □

Thus, we have completed the proof that $\mathcal{VS}^{\mathbb{Z}}(\mathcal{L})$ is closed under the 5 usual set operations stated above.

4 Query Plans and Algebraic Operations

By the definition (Definition 3) of boolean inductive queries, the solution set $Th(Q, \mathcal{D}, \mathcal{L})$ of a query Q is obtained by applying algebraic operations on the solution sets w.r.t. the underlying queries. This can be formalized using the notion of a query plan.

Definition 15. *A query plan is a boolean formula with some of its subqueries marked using $\underbrace{\ \ }$. Furthermore, all marked subqueries are the conjunction of a monotonic and an anti-monotonic subquery.*

[1] It is also anti-monotonic.

Example 16. Consider the query $Q = (a_1 \wedge m_1) \vee (a_1 \wedge m_2) \vee (a_2 \wedge m_1) \vee (a_2 \wedge m_2)$ where the a_i and m_i are anti-monotonic (resp. monotonic) predicates. The solution set of this query can be obtained by first computing the solution sets to the queries $(a_1 \wedge m_1)$, $(a_1 \wedge m_2)$, $(a_2 \wedge m_1)$ and $(a_2 \wedge m_2)$ and then taking their union. This way of computing $Th(Q, \mathcal{D}, \mathcal{L})$ corresponds to the plan

$$\underbrace{(a_1 \wedge m_1)} \vee \underbrace{(a_1 \wedge m_2)} \vee \underbrace{(a_2 \wedge m_1)} \vee \underbrace{(a_2 \wedge m_2)}$$

Alternatively, one could rewrite the query Q into $\underbrace{(a_1 \vee a_2) \wedge (m_1 \vee m_2)}$ and obtain the result as indicated in using one call to a conjunctive query solver (i.e. an algorithm or system that computes the set of all solutions to a conjunctive query). Intermediate forms are also possible.

For any inductive query Q, one can now construct a variety of different query plans by annotating queries that are logically equivalent to Q. The question then arises as to which query plan is optimal, in the sense that the resources (i.e. memory and cpu-time) needed for computing its solution set are as small as possible. A general approach to this problem would involve the use of cost estimates that for each call to a conjunctive solver and operation. One example of a cost function for a call to a conjunctive solver could be *Expected Number of Scans of Data* $\times$ *Size of Data Set*. Another one could be the *Expected Number of Covers Tests*. De Raedt *et al.* have studied (and solved) the query optimization problem under the assumption that each call to a conjunctive solver has unit cost and that the only set operation allowed is union. Under this assumption, decomposing a query Q into k subqueries of the form $Q_{a,i} \wedge Q_{m,i}$ with ($Q_{a,i}$ anti-monotonic and $Q_{m,i}$ monotonic) and $\dim(Q) = k$ is an optimal strategy. In this paper, we will leave open the challenging question as to which cost-estimates to use in practice. However, what should be clear is that given such cost-estimates, one could optimize inductive queries by constructing all possible query plans and then selecting the best one. This is effectively an optimization problem, not unlike the query optimization problem in relational databases.

4.1 Answering Interactive Queries using Incremental Update Techniques

The optimization problem becomes even more interesting in the light of interactive querying sessions [13], which should be quite common when working with inductive databases. In such sessions, one typically submits a rough query to get some insight in the domain, and when the results of this query are available, the user studies the results and refines the query. This often goes through a few iterations until the desired results are obtained.

These interactive sessions are again similar in spirit to those in traditional databases, where the results of intermediate queries are often cached for optimizing later queries. Indeed, consider that for the above example, we would already have the result of the query $(a_1 \vee a_2) \wedge m_1$. In this case, one could actually employ

the following query plan:

$$\underline{((a_1 \vee a_2) \wedge m_1)} \vee \underbrace{((a_1 \vee a_2) \wedge m_2)}$$

where $\underline{\cdots}$ denotes a query for which the solution is readily available. This query plan employs a union and could save a lot of time when m_1 is a minimum frequency query in a large database.

Even when the refined query Q_1 is not related the original one Q_0 via Boolean algebra, there is still a possibility for reusing the cached results $VS(Q_0)$ to obtain the new solution $VS(Q_1)$ efficiently.[2] One such situation occurs with the minimum frequency constraints. For example, $Q_0 = \mathsf{min\_freq}(\varphi, \mathcal{D}, \theta_1)$ and $Q_1 = \mathsf{min\_freq}(\varphi, \mathcal{D}, \theta_2) \wedge \mathsf{length\_at\_least}(\varphi, 3)$. When $\theta_2 \geq \theta_1$, then $VS(Q_1) \subseteq VS(Q_0)$. So, we can find out the answer $VS(Q_1)$ by simply performing a filtering operation on $VS(Q_0)$. If the frequencies of all patterns in $VS(Q_0)$ were stored with the results, we can even do this filtering without scanning the database! On the other hand, when $\theta_2 \leq \theta_1$, a simple filtering does not work. In this case, we can make use of an incremental update algorithm [14, 15], with adaptations, to efficiently compute $VS(Q_1)$ by using the cached results of $VS(Q_0)$.

The problem boils down to recognizing the parts of queries Q_0 and Q_1 which are related this way. For simple queries, such recognition is simple. However, for more complicated queries, it becomes challenging to recognize such related parts of the queries. Our algebraic framework allows one to break down a complicated query into a disjunctive normal form, making it easier to recognize such related parts for optimization. Moreover, under this algebraic framework, it is possible to rewrite the queries Q_0 and Q_1 into many equivalent forms, and we can then devise algorithms to recognize the related parts in these forms. This opens up another dimension of opportunities for optimizations in data mining.

5 Generalized Version Space Trees

We have extended our previous data structure "Version Space Trees" [3] for mining $\mathcal{VS}^1$ of strings to handle the general case of $\mathcal{VS}^{\mathbb{Z}}$. The extended data structure is called *Generalized Version Space Trees* (GVS Tree).

The GVS Tree maintains a set of strings which are patterns we are discovering from the database. Each such string is represented by a node in the tree. For any given node n, we denote the string pattern it represents by $s(n)$. The organization of the tree is based on suffix tries. A *suffix trie* is a trie with the following properties:

- For each node n in the trie, and for each suffix t of $s(n)$, there is also a node n' in the trie representing t, i.e. $t = s(n')$.
- Each node n has as a *suffix link*, $\mathit{suffix}(n) = n'$, such that $s(n')$ is obtained from $s(n)$ by dropping the first character. The root node is special because it represents ϵ, which has no suffixes. We define $\mathit{suffix}(root) = \Omega$, where Ω denotes a unique fake node.

[2] For brevity, we write $VS(Q)$ for $Th(Q, \mathcal{D}, \mathcal{L})$.

Unlike the common approach in the literature on suffix trees [16, 17], we use suffix tries in two very different ways from the main stream. The first one is that instead of building a suffix tree on all the suffixes of a *single* string, we are indexing all the suffixes of a *set of strings patterns* for a string database $\mathcal{D}$. This means multiple strings are stored in the tree. Moreover, in parts of our algorithms, we even keep a count of occurrences of each such substring in the corresponding node.

A *labelled trie* T_f is a suffix trie where each node is labelled with either a "$\oplus$" or a "$\ominus$". We will use the $\oplus$ label to indicate nodes representing elements in $Th(Q, \mathcal{D}, \Sigma^*) \in \mathcal{VS}^{\mathbb{Z}}(\Sigma^*)$ and $\ominus$ for those that are not. In our previous publication [3], the VS Tree had a restriction that there can be at most one sign change the root to any leave. This is because VS Tree was designed to model sets in $\mathcal{VS}^1$ only. As a generalization in this current work, we have removed this restriction, allowing complete freedom on the assignment of the labels "$\oplus$" or a "$\ominus$" to any node. As a result, a GVS Tree can represent sets of string patterns in $\mathcal{VS}^{\mathbb{Z}}$. The usual set operations can be performed by manipulating the labels on the nodes of GVS Trees, which we will come to in Sect. 5.1.

GVS Tree T is a labelled trie that represents a generalized version space $V \in \mathcal{VS}^{\mathbb{Z}}(\Sigma^*)$. In other words, $V = \{\varphi \mid \exists \text{ node } n \in T : \varphi = s(n) \wedge n \text{ is labelled } \oplus\}$. We have observed that if $\dim(V) = n$, then there exists a path (exploiting both child and suffix links, and ignoring the link directions) in T with alternating signs so that the number of sign changes from $\ominus$ to $\oplus$ is n (Lemma 8).

Fig. 1 shows a labelled GVS Tree for the set $Th(Q, \mathcal{D}, \Sigma^*)$ where $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}\}$ and $Q(\varphi, \mathcal{D}) = (\mathtt{bc} \preceq \varphi \preceq \mathtt{abcd}) \vee (\mathtt{a} \preceq \varphi \preceq \mathtt{acd})$. The dashed, curved arrows show the suffix links. The suffix links of the nodes immediate below the root node all points back to the root node, and are omitted for clarity. The $\oplus$ nodes represent the seven members of the GVS, namely `a`, `abc`, `abcd`, `ac`, `acd`, `bc` and `bcd`. Note that Q above is already in a minimal disjunctive normal form. So, the GVS has a dimension of 2, and the path through the nodes representing ϵ, `a`, `ab`, `abc` with exactly 2 sign changes.

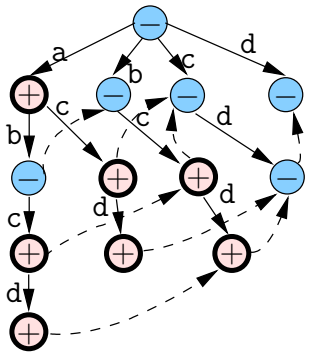


Fig. 1. An example of GVS Tree

An important property of the GVS Tree is that checking for membership is very efficient. Given any string $\varphi \in \Sigma^*$, we just need to follow the symbols on φ and descend through the tree accordingly. We will then end up at a node n so that $s(n) = \varphi$. This node has an $\oplus$ mark if and only if φ is in the GVS. The time complexity is $O(|\varphi|)$. The space complexity of a GVS Tree and the time required to build it is the same of that of suffix tries—quadratic in the size of the input strings.

5.1 Algorithm TreeMerge

The TreeMerge algorithm is basically an algorithm for merging two ordinary trees. However, we have to combine the flags ($\ominus$ or $\oplus$) from both trees, too. The combining algorithm is presented in the pseudocode in Algorithm 1 as an abstract Boolean operation $\odot$. When the operation is "and", the flags are combined using conjunction during the merging ($\ominus$ is interpreted as "false" while $\oplus$ is treated as "true"), and hence the TreeMerge algorithm will compute the intersection of the represented GVSes. When $\odot = \vee$, we get the union operation. When $x \odot y \equiv \neg(x \rightarrow y)$, we get the set difference operation.

In the algorithm, the function *root_or_negative*(T) returns the root node of a tree T if T is non-empty, or a node with label $\ominus$ and no children if T is empty. Function *child*(n, σ) returns the child node of n on the child link labelled σ, where $\sigma \in \Sigma$. If there is no such child, *NULL* is returned. The function *tree_with_root*(n) returns a GVS Tree whose root node is n, or T_{empty} if n is *NULL*. T_{empty} denotes an empty GVS Tree.

Algorithm 1 TreeMerge

Require:
/* Input: T_1, T_2: two GVS Trees $\odot$: a binary, boolean operator */
/* Output: T' = The resulting of merging T_1 and T_2, so that the flag on each node in T' is the result of applying $\odot$ to the corresponding nodes on T_1 and T_2. */

if $T_1 = T_{\text{empty}} \wedge T_2 = T_{\text{empty}}$ **then**
  return T_{empty}
$r_1 \leftarrow$ *root_or_negative*(T_1); $r_2 \leftarrow$ *root_or_negative*(T_2)
Create new tree T' with root node r'.
label(r') $\leftarrow$ *label*(r_1) $\odot$ *label*(r_2)
for all $\sigma \in \Sigma$ **do**
  $c_1 \leftarrow$ *child*(r_1, σ); $c_2 \leftarrow$ *child*(r_2, σ)
  if $c_1 \neq NULL \vee c_2 \neq NULL$ **then**
    $T_{c_1} \leftarrow$ *tree_with_root*(c_1); $T_{c_2} \leftarrow$ *tree_with_root*(c_2)
    $c' \leftarrow$ *root_or_negative*(TreeMerge($T_{c_1}, T_{c_2}, \odot$)) /* Recursively */
    add c' to r' as a child node along link σ
return T'

It should be emphasized that Algorithm 1 is just a pseudo code. In practice, much optimization can be introduced by specializing the code for each particular

Boolean operation. For instance, when $\odot = \wedge$ and a certain branch in T_1 is empty, there is no need to look at a corresponding branch in T_2.

6 Preliminary Experiments

We have implemented the algorithm VST [3] that computes the results of conjunctive queries (with dimension 1) and TreeMerge (see Sect. 5.1) in C and performed some experiments on a PC computer with a Pentium-4 2.8GMHz processor, 2GB main memory, and running Linux operating system (kernel 2.4.19, glibc 2.2.5). The former is used as our conjunctive query solver (see Sect. 4) while the latter is for performing set operations on pattern sets. Our implementation supports as primitives the two predicates min_freq and max_freq as given in Sect. 2.

6.1 The Databases

Two databases were used in the experiments. The first one is a command history collected from 168 Unix users over a period of time. [18] The users are divided into four groups: computer scientists, experienced programmers, novice programmers and non-programmers. The corresponding data subsets are denoted "sci", "exp", "nov" and "non", respectively. Each group has a number of users. When a user accesses the Unix system, he first logs in, then types in a sequence of commands, and finally logs out. Each command is taken as a symbol in the database, The sequence of commands from log in to log out constitutes a login session, which is mapped to a string in our experiment. The alphabet is the set of available commands. Each user contributes to many login sessions in the database. Table 1(a) gives some summary data on the database.

Table 1. Summary statistics of the databases

(a) Unix command database (b) Yeast database

Subset	no. of users	number of sequences
nov	55	5164
exp	36	3859
non	25	1906
sci	52	7751

Subset	number of sequences
cat30	209
cat40	2256
cat2	252

The second database is a collection of ORFs (open reading frames) of yeast, cf. [19]. The alphabet is the set of 20 amino acids. Out of these 20 symbols, every ORF (encoding for a protein) can be represented. Our database contains 6354 such sequences. Each sequence has a length between 8 and 4910. In addition, each sequence is associated with one or more functional categories (cat30: "Control of cellular organization", cat40: "Subcellular localisation", cat2: "Energy", etc.).

As preprocessing, we have separated the sequences into groups according to their functional categories. When a sequence has more than one category, it appears in multiple groups. However, within each group, a sequence appears at most once. Three of these groups were used in the experiments. They are given in Table 1(b).

6.2 The Queries

For both databases, we used a query of the following form and ran our programs to mine the patterns satisfying the constraints.

$$Q = (Q_1 \vee Q_2) \wedge Q_A$$

where

$$Q_1 = \mathsf{min\_freq}(\varphi, D_1, |D_1| \times \theta_1) \wedge \mathsf{length\_at\_least}(\varphi, 3)$$
$$Q_2 = \mathsf{min\_freq}(\varphi, D_2, |D_2| \times \theta_2) \wedge \mathsf{length\_at\_least}(\varphi, 2)$$

and Q_A is an anti-monotonic constraint. Here, D_1 and D_2 are subsets of the database being used, θ_1 and θ_2 are the corresponding minimum frequency thresholds, also set to 10% of the number of sequences in the subsets, and Q_A is an anti-monotonic constraint.

Note that Q is neither monotonic nor anti-monotonic. So, the solution set is in $\mathcal{VS}^{\mathbb{Z}}$. However, Q_1 and Q_2 are each a conjunction of an anti-monotonic constraint (minimum frequency) and a monotonic one (minimum length). So, $\dim(Q_1) = \dim(Q_2) = \dim(Q_A) = 1$. Thus, one (straight-forward) strategy of find the set of patterns satisfying Q is to use the query plan

$$\Big(\underbrace{Q_1} \vee \underbrace{Q_2}\Big) \wedge \underbrace{Q_A}$$

This involves 3 invocations of our frequent pattern VST.

However, the VST algorithm (or any other frequent pattern discovery algorithm such as Apriori [20]) is the most time-consuming part of the whole processes. We would like to minimize this cost. This is possible now with our algebra on generalized version spaces and theorems on the dimension. By Theorem 12, we know that $\dim(Q_1 \vee Q_2) \leq \dim(Q_1) + \dim(Q_2) = 1 + 1 = 2$. Now, applying Theorem 11, we have $\dim(Q) \leq \dim(Q_1 \vee Q_2) + \dim(Q_A) - 1 \leq 2 + 1 - 1 = 2$. Thus, Q has a dimension of at most 2. This means it is possible to express Q as the union of two version spaces.

Indeed, we can obtain a different query plan for Q, which we denote by Q':

$$Q' = \underbrace{Q'_1} \vee \underbrace{Q'_2}$$

where

$$Q'_1 = Q_1 \wedge Q_A$$
$$Q'_2 = Q_2 \wedge Q_A$$

This query plan involves only 2 invocations of algorithm VST. It is thus expected to be faster. Moreover, having pushed the anti-monotonic constraint Q_A deeper

into the query evaluation, we expect the levelwise algorithm VST to prune more effectively.

To verify this, we have run experiments on the two databases with the values for D_1, D_2 and Q_A as shown in Table 2. On the unix command database, this query translates to "What sequences of unix commands are used often by experienced programmers with a length of at least 3 or by computer scientists with a length of at least 2, and are also frequently used by the other two groups of users?". With our algebraic framework, it is possible to perform data mining with such complicated constraints. The query for the protein database translates to "What amino acid sequences occur frequently in function category 'cat30' with a length of at least 3 or in function category 'cat40' with a length of at least 2, and is at the same time frequent among the function category 'cat2'?".

Table 2. Details of queries used for the experiments

	Unix command database	Yeast database
D_1	experienced programmers	cat30
D_2	computer scientists	cat40
θ_1	10%	20%
θ_2	10%	20%
Q_A	$\mathsf{min\_freq}(\varphi, \text{non}, \lvert\text{non}\rvert \times 10\%)$ $\wedge \mathsf{min\_freq}(\varphi, \text{nov}, \lvert\text{nov}\rvert \times 10\%)$	$\mathsf{min\_freq}(\varphi, \text{cat2}, \lvert\text{cat2}\rvert \times 20\%)$

6.3 Results

Performance The queries Q and Q' are evaluated as described above using our implementation of the VST and TreeMerge algorithms. For each database the resulting patterns for both queries are compared and found to be identical. This verifies the correctness of our theory and implementation.

The time taken are noted and given in Table 3. With the unix command database, it took 2.15 seconds to evaluate the query as Q and only 1.60 seconds to evaluate as Q'. With the yeast database, it took 4.03 and 3.65 seconds, respectively. It is thus 9–26% faster to use strategy Q' than Q to find out the set of patterns. The table also shows a breakdown of the time taken for evaluating the queries Q_1, Q_2, Q_A, Q'_1 and Q'_2. The pattern sets for these are all in $\mathcal{VS}^1$, and are computed by the VST algorithm. It should be noted that the time taken for the TreeMerge algorithm is negligible (less than 1 ms). This confirms our claim that invocations of algorithm VST is the most time-consuming part of the whole process.

Another important observation in Table 3 is the number of patterns found for each query strategy and subqueries. In query strategy Q', the constraint Q_A is pushed down to the subqueries Q'_1 and Q'_2, effectively pruning the number

Table 3. Experimental Results

Query	No. of patterns		Time (sec.)		Heap memory (bytes)	
Strategy	Unix	Yeast	Unix	Yeast	Unix	Yeast
Q (total)	41	404	2.15	4.03	128619	119988
Q_1	110	638	0.74	0.42		
Q_2	212	122	1.18	3.34		
Q_A	67	434	0.23	0.27		
Q' (total)	41	404	1.60	3.65	66740	106476
Q'_1	16	403	0.65	0.55		
Q'_2	40	38	0.95	3.10		

of patterns that needs to be processed by the programs. This accounts for the improved speed and memory usage.

Memory Footprint Not only is time saved, but also is memory more efficiently used when we use strategy Q' instead of Q to find out the set of patterns in question. The amount of heap memory used by our programs were recorded. The maximum amount of heap memory usage is shown in Table 3. Using query evaluation strategy Q', we save 11–48% of memory. Thus, it saves both time and memory to evaluate the query using Q'.

7 Conclusions

We have generalized the notion of convex sets or version spaces to represent sets of higher dimensions. These generalized version spaces are useful for representing the solution sets to boolean inductive queries. Furthermore, we have studied the effect of algebraic operations on such generalized version spaces and shown that these generalized version spaces are closed under the set operations. This generalizes Hirsh's results on traditional version spaces (sets of dimension 1).

We have also shown how the resulting algebraic framework can be employed for query planning and optimization. The framework has been implemented for the pattern domain of strings and experimental results that illustrate the use of the framework have been presented.

Nevertheless, there are many remaining opportunities for further research. Most important is the development of effective and realistic cost functions for inductive query evaluation and their use in query optimization.

References

1. Mannila, H., Toivonen, H.: Levelwise search and borders of theories in knowledge discovery. Data Mining and Knowledge Discovery **1** (1997) 241–258
2. De Raedt, L.: A perspective on inductive databases. SIGKDD Explorations: Newsletter of the Special Interest Group on Knowledge Discovery and Data Mining, ACM **4** (2003) 69–77

3. De Raedt, L., Jaeger, M., Lee, S.D., Mannila, H.: A theory of inductive query answering (extended abstract). In: Proc. The 2002 IEEE International Conference on Data Mining (ICDM'02), Maebashi, Japan (2002) 123–130
4. Mitchell, T.M.: Generalization as search. Artificial Intelligence **18** (1982) 203–226
5. De Raedt, L., Kramer, S.: The levelwise version space algorithm and its application to molecular fragment finding. In: IJCAI01: Seventeenth International Joint Conference on Artificial Intelligence. (2001)
6. Kramer, S., De Raedt, L., Helma, C.: Molecular feature mining in hiv data. In: KDD-2001: The Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Association for Computing Machinery (2001) ISBN: 158113391X.
7. Bucila, C., Gehrke, J., Kifer, D., White, W.: DualMiner: A dual-pruning algorithm for itemsets with constraints. In: Proceedings of The Eight ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Edmonton, Alberta, Canada (2002)
8. Boulicaut, J.F.: (Habilitation thesis, 2001)
9. Hirsh, H.: Generalizing version spaces. Machine Learning **17** (1994) 5–46
10. Hirsh, H.: Theoretical underpinnings of version spaces. In: Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI91), Morgan Kaufmann Publishers (1991) 665–670
11. De Raedt, L., Jaeger, M., Lee, S.D., Mannila, H.: A theory of inductive query answering. (2003) (planned for submission to a journal).
12. Fischer, J., De Raedt, L.: Towards optimizing conjunctive inductive queries. In: Proceedings of The Third IEEE International Conference on Data Mining (ICDM'03), Melbourne, Florida, USA (2003) Submission, under review.
13. Baralis, E., Psaila, G.: Incremental refinement of mining queries. In Mohania, M.K., Tjoa, A.M., eds.: Proceedings of the First International Conference on Data Warehousing and Knowledge Discovery (DaWaK'99). Volume 1676 of Lecture Notes in Computer Science., Florence, Italy, Springer (1999) 173–182
14. Cheung, D.W.L., Lee, S.D., Kao, B.: A general incremental technique for maintaining discovered association rules. In: Proceedings of the Fifth International Conference on Database Systems for Advanced Applications, Melbourne, Australia (1997) 185–194
15. Lee, S.D., Cheung, D.: 8. In: Maintenance of Discovered Association Rules. Volume 600 of The Kluwer International Series in Engineering and Computer Science. Kluwer Academic Publishers, Boston (2000) ISBN-0-7923-7243-3.
16. Ukkonen, E.: On-line construction of suffix trees. Algorithmica **14** (1995) 249–260
17. Weiner, P.: Linear pattern matching algorithm. In: Proc. 14 IEEE Symposium on Switching and Automata Theory. (1973) 1–11
18. Greenberg, S.: Using unix: Collected traces of 168 users. Research Report 88/333/45, Department of Computer Science, University of Calgary, Calgary, Canada. (1988)
19. Clare, A., King, R.D.: Machine learning of functional class from phenotype data. Bioinformatics **18** (2002) 160–166
20. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules. In Bocca, J.B., Jarke, M., Zaniolo, C., eds.: Proc. 20th Int. Conf. Very Large Data Bases, VLDB, Morgan Kaufmann (1994) 487–499

Finding All Occurring Sets of Interest

Taneli Mielikäinen

HIIT Basic Research Unit
Department of Computer Science
University of Helsinki, Finland
Taneli.Mielikainen@cs.helsinki.fi

Abstract. In this paper we examine the problem of mining all occurring sets of interest. We define what they are, sketch some applications, describe streaming algorithms for the problem and analyze their computational complexity. We also study alternative representations for the occurring sets of interest and evaluate some of them experimentally.

1 Introduction

Finding all sets that occur frequently in a given data set has been a very popular research topic in data mining for years [1,2,3,4,5]. This *frequent set mining problem* can be stated as follows: given a finite set R of *attributes*, a sequence $d = d_1, \ldots, d_n$ of its subsets $d_i \in 2^R, 1 \leq i \leq n$, and a threshold value $\sigma \in [0, 1]$, find all σ-frequent sets in d, i.e., find all subsets X of R such that the frequency

$$fr(X) = fr(X, d) = \frac{|\{i : X \subseteq d_i, 1 \leq i \leq n\}|}{n}$$

of X is at least σ. The collection of σ-frequent sets in d is denoted by

$$\mathcal{F}_{\sigma,d} = \{X \subseteq R : fr(X, d) \geq \sigma\}.$$

The set of different sets in d is denoted by $\mathcal{D} = \{d_i : 1 \leq i \leq n\}$ and the number of (exact) occurrences of set X in d is denoted by $occ(X, d) = |\{i : X = d_i\}|$. The sequence d is sometimes called a *database*. Then the elements d_i of d are called *transactions* or *rows*.

One practical reason why the frequencies of only the σ-frequent sets are computed instead of the frequencies of the whole set collection 2^R, is that 2^R contains $2^{|R|}$ subsets which is too large to be practical already for very small numbers of attributes. Also, the frequent sets are presumed to contain the most relevant sets in 2^R. However, it is not always easy to find a good minimum frequency threshold σ such that $\mathcal{F}_{\sigma,d}$ would contain most of the relevant sets and only few irrelevant ones. It might be even possible that such threshold does not exist at all. This is the case, for example, when some combinations of attributes are are not relevant no matter of their frequencies.

These observations suggest to study a task complementary to the frequent set mining, namely mining all sets occurring in d that are contained in some of

the given subsets of attributes. The problem can be formulated more precisely as follows: given a finite set R of attributes, a sequence $d = d_1, \ldots, d_n$ of its subsets, and collection $\mathcal{S}$ of subsets of R, find all *occurring sets of interest* in d, i.e., find all subsets of R that are contained in at least one set in $\mathcal{S}$. Analogously to the set of frequent sets, the set of occurring sets of interest is denoted by

$$\mathcal{I}_{\mathcal{S},d} = \{X \subseteq Y \cap Z : Y \in \mathcal{S}, Z \in \mathcal{D}\}.$$

The set $\mathcal{I}_{\mathcal{S},d}$ can be computed by first generating all subsets of the sets in $\mathcal{S}$ and then counting their frequencies in d. As the collection $\mathcal{F}_{\sigma,d}$, also the collection $\mathcal{I}_{\mathcal{S},d}$ can be very redundant. The redundancy of frequent sets has motivated several studies of *condensed (or concise) representations* of set collections, i.e., subsets of the set collection (and their frequencies) from which all the other sets (and their frequencies) in the collection can be inferred. Some of the condensed representations, e.g. closed sets [6,7,8,9], disjunction-free sets [10] and their generalizations [11,12,13,14], represent both the frequent sets and their exact frequencies while some other allow approximate representations of frequencies [15,16,17,18,19,20,21] or determine just the collection of frequent sets without the actual frequencies [22,23,24,25]. These representations are usually based on the frequencies of the frequent sets and they can be significantly smaller than the collection of all frequent sets. The condensed representations for frequent sets can be adapted to the occurring sets of interest.

In this paper we shall use two condensed representations called *maximal sets* and *closed sets*. The maximal sets suffice to determine uniquely the collection of frequent sets.

Definition 1. *A set $X \in \mathcal{C}$ is maximal in a set collection $\mathcal{C}$ iff the collection does not contain any of its supersets, i.e., iff $Y \supset X \Rightarrow Y \notin \mathcal{C}$. The set of maximal sets in a collection $\mathcal{C}$ is denoted by $max(\mathcal{C})$.*

If we are interested also in the actual frequencies of the frequent sets, we can determine maximal sets using each frequency in $\{fr(X,d) : X \in \mathcal{F}_{\sigma,d}\}$ as the minimum frequency threshold. These sets together form the collection of closed sets in $\mathcal{F}_{\sigma,d}$.

Definition 2. *A set $X \in \mathcal{C}$ is closed in a set collection $\mathcal{C}$ (w.r.t. d) iff it has no supersets in $\mathcal{S}$ with the same frequency, i.e., iff $X \subset Y \in \mathcal{C} \Rightarrow fr(X,d) > fr(Y,d)$.*

From the above mentioned condensed representations one can derive another application of the occurring sets of interest, in addition of being an alternative to minimum frequency thresholds for the set collections: the occurring sets of interest can be used to refine already computed coarse condensed representations. For example, the closed frequent sets can be computed from the maximal frequent sets and the data set. This kind of resource-aware mining with additional information might be useful e.g. in ubiquitous computing.

We have developed some streaming algorithms for mining all occurring sets of interest, i.e., algorithms that find all occurring sets of interest without storing

the sequence d. (For a short introduction to data streams, see e.g. [26].) Thus the collection of the occurring sets of interest can be used also as a summary of a data stream by choosing the set $\mathcal{S}$ to be a collection of few small random subsets of R and maintaining the occurring sets of interest for the set $\mathcal{S}$. This approach does not require any data set dependent parameters as the minimum frequency threshold in frequent set mining.

The rest of the paper is organized as follows. In Section 2 we consider the computational problem of finding all occurring closed sets of interest. In Section 3 we examine alternative representations for the occurring sets of interest. Section 4 concludes the paper.

2 Mining Closed Occurring Sets of Interest

In this section we consider the problem of computing all closed (occurring) sets of interest. We describe several algorithms and analyze their computational complexity. Also, we propose a new data structure that might have some interest of its own. Let us first show a useful lemma about closed sets:

Lemma 1. *The set $X \subseteq R$ is closed in $2^R = \{Y \subseteq R\}$ iff $X = \bigcap_{i \in I} d_i$ for some $I \subseteq [n] = \{1, \ldots, n\}$.*

Proof. By Definition 2, $X \subseteq R$ is closed in 2^R iff $fr(X,d) > fr(Y,d)$ for all $Y \supset X, Y \subseteq R$. If $X = \bigcap_{i \in I} d_i$ for some $I \subseteq [n]$ then for all of its supersets $Y \subseteq R$ there is $d_i, i \in I$, such that $Y \not\subseteq d_i$, i.e., $fr(X,d) > fr(Y,d)$. On the other hand, if $X \neq \bigcap_{i \in I} d_i$ for any $I \subseteq [n]$, then $X \subset \bigcap_{i \in I} d_i = Y$ for some $I \subseteq [n]$ and thus there is $Y \supset X, Y \subseteq R$, such that $fr(X,d) = fr(Y,d)$. □

Applying Lemma 1 we describe an algorithm that computes all the closed sets exhaustively and intersects them by the sets in $\mathcal{S}$:

INTERSECT-EXHAUSTIVE($\mathcal{S}, d$)

```
 1  ℐ ← ∅
 2  for each I ⊆ {1,...,n}
 3     do X ← ⋂_{i∈I} d_i
 4        for each Y in 𝒮
 5           do Z ← X ∩ Y
 6              if Z ∉ ℐ
 7                 then ℐ ← ℐ ∪ Z
 8                      supp[Z] ← 0
 9              if supp[Z] < |I|
10                 then supp[Z] ← |I|
11  for each X in ℐ
12     do fr(X) ← supp[X]/n
13  return (ℐ, fr(ℐ))
```

Let m denote $\min\{\max_{1 \leq i \leq n} |d_i|, \max_{Y \in \mathcal{S}} |Y|\}$. Then we can show that the algorithm INTERSECT-EXHAUSTIVE has the following time complexity:

Theorem 1. *Algorithm* INTERSECT-EXHAUSTIVE *finds all closed sets of interest and their frequencies in time* $\mathcal{O}(m2^n|\mathcal{S}|)$.

Proof. By Lemma 1, the algorithm computes all closed sets in 2^R and projects them by $\mathcal{S}$. Thus the set $\mathcal{I}$ the algorithm produces is equal to $\mathcal{I}_{\mathcal{S},d}$.

The algorithm computes $2^n|\mathcal{S}|$ intersections, the number of terms in the intersection is $n+1$ in the worst case and the number of comparisons needed to intersect two sets is $|R|$ in the worst case. Thus all intersections can be computed in time $\mathcal{O}(|R|n2^n|\mathcal{S}|)$.

However, by computing all intersections of k sets before the intersections of $k+1$ sets and memorizing the intersections of k sets, each intersection can be computed in time $|R|$. Furthermore, the intersection between sets X and Y can be computed in time $\mathcal{O}(\min\{|X|,|Y|\})$ by testing which elements of the smaller set are contained in the larger one. □

Unfortunately, the algorithm INTERSECT-EXHAUSTIVE is too slow. Moreover, it has to memorize the whole sequence d as the intersections of subsets have to be computed. (In fact, already computing all pairwise intersections $d_i \cap d_j, 1 \leq i < j \leq n$, would require this.) Note that because the sets in $\mathcal{S}$ should reduce the huge set of all subsets of R to a smaller collection of the occurring sets of interest, storing the sequence d itself should not be necessary.

It turns out that the computation of the intersections can reorganized to be more efficient. This can be done by noticing that all possible combinations of intersections producing some closed set X do not have to be computed: it is enough to produce each closed set once and to be able to compute its frequency. This observation can be formulated as an incremental streaming output-efficient algorithm as follows:

INTERSECT-INCREMENTAL$(d, \mathcal{S})$
1 $\mathcal{I} \leftarrow \mathcal{S}$
2 **for each** Y **in** $\mathcal{I}$
3 **do** $lastVisited[Y] \leftarrow 0$
4 **for** $i \leftarrow 1$ **to** n
5 **do for each** Y **in** $\mathcal{I}$
6 **do** $Z \leftarrow d_i \cap Y$
7 **if** $Z \notin \mathcal{I}$
8 **then** $\mathcal{I} \leftarrow \mathcal{I} \cup Z$
9 $supp[Z] \leftarrow 1$
10 $lastVisited[Y] \leftarrow i$
11 **if** $lastVisited[Y] < i$
12 **then** $supp[Z] \leftarrow supp[Z] + 1$
13 **for each** X **in** $\mathcal{I}$
14 **do if** $supp[X] = 0$
15 **then** $\mathcal{I} \leftarrow \mathcal{I} \setminus \{X\}$
16 **else** $fr(X) \leftarrow supp[X]/n$
17 **return** $(\mathcal{I}, fr(\mathcal{I}))$

Theorem 2. *The algorithm* INTERSECT-INCREMENTAL *finds all closed sets of interest and their frequencies in time* $\mathcal{O}(nm(|\mathcal{I}_{\mathcal{S},d}| + |\mathcal{S}|))$.

Proof. Let $\mathcal{I}_0, \mathcal{I}_1, \ldots, \mathcal{I}_n = \mathcal{I}$ denote the solutions found by the algorithm INTERSECT-INCREMENTAL for prefixes of d of lengths $0, 1, \ldots n$, respectively. Clearly, $\mathcal{I}_0 \subseteq \mathcal{I}_1 \subseteq \ldots \subseteq \mathcal{I}_n$. We first show that $\mathcal{I} = \mathcal{I}_{\mathcal{S},d} \cup \mathcal{S}$ by induction on the length of the sequence:

1. If $|d| = 0$, then there are no occurring sets in d and $\mathcal{I}_0 = \mathcal{S}$.
2. Assume that for some $k < n$, $\mathcal{I}_i = \mathcal{I}_{\mathcal{S},d_1 \ldots d_i} \cup \mathcal{S}$ holds for all $0 \leq i \leq k$. Then

$$\begin{aligned}\mathcal{I}_{k+1} &= \mathcal{I}_k \cup \{d_{k+1} \cap X : X \in \mathcal{I}_k\} \cup \{d_{k+1} \cap Y : Y \in \mathcal{S}\} \\ &= \left\{\bigcap_{i \in I} d_i \cap Y : Y \in \mathcal{S}, I \subseteq [k]\right\} \cup \left\{d_{i+1} \cap \bigcap_{i \in I} d_i \cap Y : Y \in \mathcal{S}, I \subseteq [k]\right\} \\ &= \left\{Y \cap \bigcap_{i \in I} d_i : Y \in \mathcal{S}, I \subseteq [k+1]\right\} = \mathcal{I}_{\mathcal{S},d_1 \ldots d_{k+1}} \cup \mathcal{S}.\end{aligned}$$

   Thus $\mathcal{I}_i = \mathcal{I}_{\mathcal{S},d_1 \ldots d_i} \cup \mathcal{S}$ holds for all $0 \leq i \leq n$. In particular, $\mathcal{I} = \mathcal{I}_n = \mathcal{I}_{\mathcal{S},d} \cup \mathcal{S}$ as claimed.

The counts of the sets in $\mathcal{S} \setminus \mathcal{I}_{\mathcal{S},d}$ remain zero and they can be removed from the collection $\mathcal{I}_{\mathcal{S},d}$ in time $\mathcal{O}(|\mathcal{I}|)$. The frequencies of the closed occurring sets of interest are correct because each $d_i, 1 \leq i \leq n$, increases the count of each $X \in \mathcal{I}_{\mathcal{S},d}$ contained in d_i exactly once.

As each set $d_i, 1 \leq i \leq n$, can intersect at most $|\mathcal{I}|$ sets in $\mathcal{I}_{\mathcal{S},d} \cup \mathcal{S}$ and one intersection can be computed in time $\mathcal{O}(m)$, the set $\mathcal{I} \setminus \mathcal{S} = \mathcal{I}_{\mathcal{S},d}$ and the frequencies of the sets $X \in \mathcal{I}_{\mathcal{S},d}$ can be computed in time $\mathcal{O}(nm|\mathcal{I}|) = \mathcal{O}(nm(|\mathcal{I}_{\mathcal{S},d}| + |\mathcal{S}|))$. □

One problem with the algorithm INTERSECT-INCREMENTAL is that it can compute the empty intersection several times for each set d_i. We attempt to avoid this problem by introducing a data structure called *skewers* that allows efficient computation of non-empty intersections.

Definition 3 (Skewers). *A skewers data structure* $\mathcal{V}$ *represents a collection of sets. Each set* X *in the skewers consists a set of attributes* $\mathcal{V}^X_X$*, a counter* $\mathcal{V}^X_{supp}$ *and the time of last visit* $\mathcal{V}^X_{time}$.

The sets in $\mathcal{V}$ *are in ascending order in their cardinality. In addition there is a skewer* $\mathcal{V}^A$ *for each attribute* $A \in R$ *consisting the sets containing* A *in the order conforming the global order of sets.*

The skewers $\mathcal{V}$ *supports operations insertion, deletion and location of a set* X *in time* $\mathcal{O}(|X| \log |\mathcal{V}|)$*. Next and previous set in each skewer* $\mathcal{V}^A$ *can be found in constant time.*

The intersection algorithm for skewers incrementally intersect the sets in the skewers data structure by scanning over the skewers. The scan over the sets is implemented by a priority queue.

```
INTERSECT-SKEWERS(d, S)
 1  I ← ∅
 2  V ← ∅
 3  for each Y ∈ S
 4     do INSERT-SKEWERS(Y)
 5  for i ← 1 to n
 6     do Q ← ∅
 7        for each A in d_i
 8           do INSERT-QUEUE(Q, V^A_head)
 9        while Q ≠ ∅
10           do X ← EXTRACT-QUEUE(Q)
11              Y ← X ∩ d_i
12              if V^Y = NIL
13                 then INSERT-SKEWERS(Y)
14                      I ← I ∪ {Y}
15              if V^Y_time < i
16                 then V^Y_supp ← V^Y_supp + 1
17                      V^Y_time ← i
18              for each A in X
19                 do INSERT-QUEUE(Q, NEXT-SKEWER(V^A))
20  for each X in I
21     do fr(X) ← V^X_supp / n
22  return (I, fr(I))
```

Theorem 3. *The algorithm* INTERSECT-SKEWERS *finds all closed sets of interest and their frequencies in time*

$$\mathcal{O}\left(\sum_{i=1}^{n}(|d_i|+\log|\mathcal{V}|)\sum_{A\in d_i}\left|\mathcal{V}^A\right|\right).$$

Proof. All closed sets of interest and their frequencies are found as the algorithm INTERSECT-SKEWERS incrementally constructs the collection of all closed sets of interest similarly to the algorithm INTERSECT-INCREMENTAL.

The number of sets intersected by d_i is bounded by $\sum_{A\in d_i}\left|\mathcal{V}^A\right|$. Each intersection can be computed in time $|d_i|$ and the set corresponding to the intersection can be found in time $\log|\mathcal{V}|$. Thus, the combined time complexity is $\mathcal{O}\left(\sum_{i=1}^{n}(|d_i|+\log|\mathcal{V}|)\sum_{A\in d_i}\left|\mathcal{V}^A\right|\right)$ as claimed. □

The time bound is quite pessimistic as the bound does not take into account e.g. that the same set is not added into the priority queue more than once per scan. Furthermore, if all data d can be stored, then the computation can be made more efficient by replacing the sequence d by the set $\mathcal{D}$ with number of occurrences $occ(X,d)$ for each $X\in\mathcal{D}$.

The algorithms can be adapted to mine the closed frequent sets instead of the closed sets of interest by maintaining upper bounds of frequencies for the closed sets and removing a closed set when it is clear that the set is infrequent.

3 Simplified Databases

It is not clear whether the closed sets of interest should be mined explicitly. For example, the closed sets are not very good as an index structure for frequency queries by subsets of R.

It is easy to see that the number of the closed sets in 2^R is at least as large as $|\mathcal{D}|$ due to the fact that each set in $\mathcal{D}$ is a closed set. Usually the number of closed sets is even higher because also all intersections $\bigcap_{X \in \mathcal{C}} X, \mathcal{C} \subseteq \mathcal{D}$, are closed sets. However, if we are considering the closed sets of interest instead of all closed sets in 2^R, then the number of different rows in the database d is not necessarily smaller: The set collection $\mathcal{S}$ can be chosen to be the collection of the maximal frequent sets. Then the set of the closed occurring sets of interest is the collection of the closed frequent sets. The collection of the closed frequent sets can be much smaller than the original database d or even smaller than the number $|\mathcal{D}|$ of different rows in d.

However, also the database d can be condensed. The simplest approach is to replace the database by the set $\mathcal{D}$ and the number of occurrences $occ(X, d)$ for each $X \in \mathcal{D}$. The number of sets in $\mathcal{D}$ can be further reduced by removing all attributes that are not present in $\mathcal{S}$. If the number of sets in $\mathcal{S}$ is very small, it can be worthwhile to store the projection of the database $\mathcal{D}$ onto Y for each $Y \in \mathcal{S}$ separately.

We tested with few data sets how the sizes of different representations of the set collection and the database relate to each other. The experiments were done using IPUMS Census data set from UCI KDD Repository[1] and Ilmo data set from the course enrollment system of Department of Computer Science, University of Helsinki. IPUMS Census consists of 88443 rows (88211 different rows) and 39954 attributes. Ilmo consists of 3505 rows (1903 different rows) and 97 attributes. We simulated different set collections $\mathcal{S}$ by computing the maximal σ-frequent sets for several minimum frequency thresholds σ. Thus in our experiments the sets in $\mathcal{S}$ correspond to the maximal σ-frequent sets.

The results are shown in Table 1 and Table 2. The column "pruned rows" corresponds to the number of different rows in d after removing the attributes that do not occur in $\mathcal{S}$. The column "projections" corresponds to the combined number of different rows in each projection of $\mathcal{D}$ onto $Y \in \mathcal{S}$. It can be seen from the tables that all of the representations have their good sides. The representation of database based on separate projections to all sets in $\mathcal{S}$ do nicely with high minimum frequency thresholds but eventually, as the minimum frequency threshold decreases, the collection of pruned database rows succeeds to be the smallest one.

In general, the small representations of the databases and occurring sets of interest seem to have some nontrivial computational problems:

1. Given a data set d and a collection $\mathcal{S}$ of subsets of R, find the smallest data set d' that agrees with d when projected onto any $Y \in \mathcal{S}$.

[1] `http://kdd.ics.uci.edu`

σ	$\mathcal{S} := max\left(\mathcal{F}_{\sigma,d}\right)$	$cl\left(\mathcal{I}_{\mathcal{S},d}\right) = cl\left(\mathcal{F}_{\sigma,d}\right)$	$\mathcal{I}_{\mathcal{S},d} = \mathcal{F}_{\sigma,d}$	pruned rows	projections
0.40	41	1517	1517	11934	595
0.38	56	2111	2111	13490	847
0.36	66	2795	2795	15424	1154
0.34	82	3975	3975	15993	1636
0.32	107	5361	5361	17777	2400
0.30	126	8205	8205	22626	3083
0.28	152	11443	11443	22763	4179
0.26	198	17503	17503	22763	5774
0.24	272	23903	23903	27429	8450
0.22	387	53203	53203	31488	12935
0.20	578	86879	86879	39730	22616
0.18	789	250441	250441	40128	37435
0.16	1082	524683	524683	44534	63784

Table 1. IPUMS Census data set

σ	$\mathcal{S} := max\left(\mathcal{F}_{\sigma,d}\right)$	$cl\left(\mathcal{I}_{\mathcal{S},d}\right) = cl\left(\mathcal{F}_{\sigma,d}\right)$	$\mathcal{I}_{\mathcal{S},d} = \mathcal{F}_{\sigma,d}$	pruned rows	projections
0.040	102	286	286	1546	978
0.038	113	355	355	1546	1328
0.036	132	430	430	1546	1878
0.034	123	512	512	1581	1713
0.032	123	594	594	1620	1958
0.030	157	691	691	1651	2645
0.028	190	847	847	1651	3519
0.026	226	1090	1095	1693	4814
0.024	260	1363	1378	1693	6192
0.022	292	1741	1771	1693	8380
0.020	359	2264	2348	1699	11605
0.018	436	3017	3226	1714	15109
0.016	565	4078	4519	1723	22044

Table 2. Ilmo data set

2. Given a data set d and a collection $\mathcal{S}$ of subsets of R, find a collection of data sets $d^1, \ldots, d^k$ with the smallest combined number of different rows and a mapping $f : \mathcal{S} \to [k]$ such that $d^{f(Y)}$ agrees with d when projected onto any $Y \in \mathcal{S}$.

The second problem can be attempted to solve by the following heuristic:

1. Compute projections d^Y of the database d onto each $Y \in \mathcal{S}$.
2. Find the pair of projections d^Y and d^Z such that $\Delta = \left|d^Y\right| + \left|d^Z\right| - \left|d^{Y \cup Z}\right|$ is largest.
3. If $\Delta > 0$, then replace d^Y and d^Z by $d^{Y \cup Z}$ and go to step 2.

The algorithm can be implemented in time $\mathcal{O}\left(|R|\,|\mathcal{D}|\,|\mathcal{S}|^2\right)$ where $R = \bigcup_{Y \in \mathcal{S}} Y$. The solution found by the heuristic is at least as good as projecting the database the separately onto each set in $\mathcal{S}$.

4 Conclusions

In this paper we have studied the problem of finding sets X contained in a set d_i of a data set d such that $X \subseteq Y$ for some $Y \in \mathcal{S}$. We have proposed streaming algorithms for the problem and discussed about other possible representations for the occurring sets of interest. There are some interesting open problems:

- How the closed sets of interest with $\mathcal{S}$ as a collection of random projections could be used as a useful summary for a data stream?
- Can the algorithms used to speed up the frequent set mining in conjunction with some efficient implementation of a maximal set mining algorithm?
- Where the skewers data structure could be applied?
- Is there some clear relationship between the overlap of sets in $\mathcal{S}$, the size of the collection $\mathcal{I}_{\mathcal{S},d}$ and the size of d?
- How a database can be transformed into a smallest form that can be used to answer to certain queries efficiently and (approximately) correctly?

References

1. Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., Verkamo, A.I.: Fast discovery of association rules. In Fayyad, U.M., Piatetsky-Shapiro, G., Smyth, P., Uthurusamy, R., eds.: Advances in Knowledge Discovery and Data Mining. AAAI/MIT Press (1996) 307–328
2. Goethals, B.: Survey on frequent pattern mining. Manuscript (2003)
3. Hand, D.J.: Pattern detection and discovery. In Hand, D., Adams, N., Bolton, R., eds.: Pattern Detection and Discovery. Volume 2447 of LNAI., Springer-Verlag (2002) 1–12
4. Hipp, J., Güntzer, U., Nakhaeizadeh, G.: Algorithms for association rule mining – a general survey and comparison. SIGKDD Explorations **1** (2000) 58–64
5. Mannila, H.: Local and global methods in data mining: Basic techniques and open problems. In Widmayer, P., Triguero, F., Morales, R., Hennessy, M., Eidenbenz, S., Conejo, R., eds.: Automata, Languages and Programming. Volume 2380 of LNCS., Springer-Verlag (2002) 57–68
6. Pasquier, N., Bastide, Y., Taouil, R., Lakhal, L.: Discovering frequent closed itemsets for association rules. In Beeri, C., Buneman, P., eds.: Database Theory - ICDT'99. Volume 1540 of LNCS., Springer-Verlag (1999) 398–416
7. Pei, J., Han, J., Mao, T.: CLOSET: An efficient algorithm for mining frequent closed itemsets. In Gunopulos, D., Rastogi, R., eds.: ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery. (2000) 21–30
8. Stumme, G., Taouil, R., Bastide, Y., Pasquier, N., Lakhal, L.: Computing iceberg concept lattices with Titanic. Data & Knowledge Engineering **42** (2002) 189–222
9. Zaki, M.J., Hsiao, C.J.: CHARM: An efficient algorithms for closed itemset mining. In Grossman, R., Han, J., Kumar, V., Mannila, H., Motwani, R., eds.: Proceedings of the Second SIAM International Conference on Data Mining, SIAM (2002)
10. Bykowski, A., Rigotti, C.: A condensed representation to find frequent patterns. In: Proceedings of the Twenteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, ACM (2001)

11. Calders, T., Goethals, B.: Mining all non-derivable frequent itemsets. In Elomaa, T., Mannila, H., Toivonen, H., eds.: Principles of Data Mining and Knowledge Discovery. Volume 2431 of LNAI., Springer-Verlag (2002) 74–865
12. Calders, T., Goethals, B.: Minimal k-free representations of frequent sets. In Lavrac, N., Gamberger, D., Todorovski, L., Blockeel, H., eds.: Principles of Knowledge Discovery and Data Mining. LNAI, Springer-Verlag (2003)
13. Kryszkiewicz, M.: Concise representation of frequent patterns based on disjunction-free generators. In Cercone, N., Lin, T.Y., Wu, X., eds.: Proceedings of the 2001 IEEE International Conference on Data Mining, IEEE Computer Society (2001) 305–312
14. Kryszkiewicz, M., Gajek, M.: Concise representation of frequent patterns based on generalized disjunction-free generators. In Chen, M.S., Yu, P., Liu, B., eds.: Advances in Knowledge Discovery and Data Mining. Volume 2336 of LNAI., Springer-Verlag (2002) 159 – 171
15. Bouliçaut, J.F., Bykowski, A.: Frequent closures as a concise representation for binary data mining. In Terano, T., Liu, H., Chen, A.L.P., eds.: Knowledge Discovery and Data Mining. Volume 1805 of LNAI., Springer-Verlag (2000) 62–73
16. Bouliçaut, J.F., Bykowski, A., Rigotti, C.: Free-sets: a condensed representation of Boolean data for the approximation of frequency queries. Data Mining and Knowledge Discovery **7** (2003) 5–22
17. Geerts, F., Goethals, B., Mielikäinen, T.: What you store is what you get (extended abstract). In: 2nd International Workshop on Knowledge Discovery in Inductive Databases. (2003)
18. Mielikäinen, T.: Frequency-based views to pattern collections. In: IFIP/SIAM Workshop on Discrete Mathematics and Data Mining. (2003)
19. Mielikäinen, T., Mannila, H.: The pattern ordering problem. In Lavrac, N., Gamberger, D., Todorovski, L., Blockeel, H., eds.: Principles of Knowledge Discovery and Data Mining. LNAI, Springer-Verlag (2003)
20. Pavlov, D., Mannila, H., Smyth, P.: Beyond independence: probabilistic methods for query approximation on binary transaction data. IEEE Transactions on Data and Knowledge Engineering (2003) To appear.
21. Pei, J., Dong, G., Zou, W., Han, J.: On computing condensed pattern bases. In: Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM 2002), 9-12 December 2002, Maebashi City, Japan, IEEE Computer Society (2002) 378–385
22. Bayardo Jr., R.J.: Efficiently mining long patterns from databases. In Laura M. Haas, A.T., ed.: SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data, ACM (1998) 85–93
23. Boros, E., Gurvich, V., Khachiyan, L., Makino, K.: On the complexity of generating maximal frequent and minimal infrequent sets. In Alt, H., Ferreira, A., eds.: STACS 2002. Volume 2285 of LNCS., Springer-Verlag (2002) 133–141
24. Gouda, K., Zaki, M.J.: Efficiently mining maximal frequent itemsets. In Cercone, N., Lin, T.Y., Wu, X., eds.: Proceedings of the 2001 IEEE International Conference on Data Mining. IEEE Computer Society (2001) 163–170
25. Gunopulos, D., Khardon, R., Mannila, H., Saluja, S., Toivonen, H., Sharma, R.S.: Discovering all most specific sentences. ACM Transactions on Database Systems **28** (2003) 140–174
26. Babcock, B., Babu, S., Datar, M., Motwani, R., Widom, J.: Models and issues in data stream systems. In Popa, L., ed.: Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, ACM (2002) 1–16

Mining concepts from large SAGE gene expression matrices

François Rioult[1], Céline Robardet[2,3], Sylvain Blachon[2], Bruno Crémilleux[1], Olivier Gandrillon[2], and Jean-François Boulicaut[3]

[1] GREYC CNRS UMR 6072, F-14032 Caen, France
Francois.Rioult@info.univ-caen.fr
[2] CGMC CNRS UMR 5534, F-69622 Villeurbanne cedex, France
gandrillon@cgmc.univ-lyon1.fr
[3] LIRIS CNRS FRE 2672, F-69621 Villeurbanne cedex, France
Jean-Francois.Boulicaut@insa-lyon.fr

Abstract. One of the crucial needs in post-genomic research is to analyze expression matrices (e.g., SAGE and microarray data) to identify a priori interesting sets of genes, e.g., sets of genes that are frequently co-regulated. Such matrices provide expression values for given biological situations (the lines) and given genes (columns). The inductive database framework enables to support knowledge discovery processes by means of sequences of queries that concerns both data processing and pattern querying (extraction, post-processing). We provide a simple formalization of a relevant pattern domain (language of patterns, evaluation functions and primitive constraints) that has been proved useful for specifying various analysis tasks. Recent algorithmic results w.r.t. the efficient evaluation (constraint-based mining) of the so-called inductive queries are emphasized and illustrated on a $90 \times 12\,636$ human SAGE expression matrix.

1 Introduction

We are now entering the post-genome era and it seems obvious that, in a near future, the critical need will not be to generate data, but to derive knowledge from huge data sets generated at very high throughput. This has been a challenge for quite some time in genomic research, and is now extending to the domain of transcriptome research, i.e., the analysis of gene expression data. Different techniques (including microarray [12] and SAGE [23]) allow to study the simultaneous expression of (tens of) thousands of genes in various biological situations. The data generated by those experiments can then be seen as expression matrices in which the expression level of genes (the columns) are recorded in various biological situations (the lines). What is presently required is to try to find out groups of co-regulated genes, also known as synexpression groups [19], which, based on the guilt by association approach, are assumed to participate in a common function, or module, within the cell. Indeed, biologist often use clustering techniques to identify sets of

genes that have similar expression profiles (see, e.g., [13]). Recently, association rule mining has been studied as a complementarity approach for the identification of a priori interesting set of gene [2]. An added-value is to provide a symbolic description that quantify the "association" between sets of genes w.r.t. to boolean expression properties (e.g., over-expression, under-expression, strong variation). Mining large gene expression matrices gives rise to new problems w.r.t. the standard application of association rule mining (e.g., for basket analysis). However, thanks to the properties of the so-called *condensed representations* and Galois connections, [22] shows that it is possible to mine *concepts* [24] in microarray data. Concept post-processing enables to perform various tasks like conceptual clustering or frequent association rule mining (over genes and/or biological situations).
The contribution of this paper is threefold. First, it describes gene expression data analysis within an inductive database approach [14, 6, 10]. For that purpose, we provide a formalization of the *pattern domain* `RNA` and discuss few evaluation functions and primitive constraints that have been proved useful. Next, recent algorithmic results w.r.t. the efficient evaluation of the so-called *inductive queries* are introduced. Finally, we discuss an original research on human SAGE data that leads to a $90 \times 27\,679$ expression matrix analysis, i.e., a quite large expression matrix w.r.t. previous work. Even though this paper does not present new biological results, the overall approach in biological terms has been already validated on a reduced set of genes [2]. We are pretty confident that given the breakthrough into extraction feasibility, biological meaning will now be extracted almost at will.
In Section 2, we define the `RNA` pattern domain and thus an inductive database approach on gene expression data analysis. In Section 3, we consider inductive query optimization issues. In Section 4 we describe the experimental validation of our approach on two matrices built from public human SAGE data. Section 5 concludes.

2 The `RNA` pattern domain

Mannila and Toivonen have formalized useful data mining tasks as follows [16]. Given, a language of patterns or models $\mathcal{L}$ to be considered, a database $\mathbf{r}$ and a selection predicate q, the aim is then to find the theory $Th(\mathcal{L}, q, \mathbf{r}) = \{\phi \in \mathcal{L} \mid q(\phi, \mathbf{r}) \text{ is true}\}$. Furthermore, it is clear that, in many situations, users are interested in *extended theories*, i.e., not only elements from $\mathcal{L}$ but also the results of some evaluation functions for these a priori interesting patterns or models (e.g., frequency, accuracy). Computing theories can be embedded into the general framework of *inductive databases* as it has been formalized in [6].
The *schema of an inductive database* is a pair $(\mathbf{R}, (\mathcal{Q}_\mathbf{R}, \mathcal{E}))$, where $\mathbf{R}$ is a database schema, $\mathcal{Q}_\mathbf{R}$ is a collection of patterns or models, $\mathcal{E}$ is a collection of evaluation functions that define pattern or model properties in the data. An instance of the schema, an *inductive database* $(\mathbf{r}, s)$ consists of a database $\mathbf{r}$ over the schema $\mathbf{R}$ and a subset $s \subseteq \mathcal{Q}_\mathbf{R}$. A typical KDD process operates on both of the components of an inductive

database. The user can select data from $\mathbf{r}$ and s remains the same. The user can also select subsets s, and $\mathbf{r}$ is not modified. Let us just consider simple typical queries, i.e., selections[1]. A data selection example is $\sigma_C(\mathbf{r}_0, s_0) = (\mathbf{r}_1, s_1)$ where $\mathbf{r}_1 = \sigma_C(\mathbf{r}_0)$ and $s_1 = s_0$. For instance, we will use this kind of query to remove some biological situations that do not verify criterion C. Any data manipulation can be performed there. A pattern selection example is $\tau_{C'}(\mathbf{r}_0, s_0) = (\mathbf{r}_2, s_2)$ where $\mathbf{r}_2 = \mathbf{r}_0$ and s_2 contains only the patterns or models that satisfy criterion C'. For instance, this kind of query can be used to select sets of genes that have some desired properties, e.g., co-regulation in at least p biological situations. Queries on inductive databases satisfy a *closure property*: queries that return data, queries that return patterns or models (data mining and post-processing queries) and queries that cross over the data and the patterns or models (post-processing queries) are all queries on inductive databases and return an inductive database. For instance, we can compute $\tau_{C'}(\sigma_C(\mathbf{r}_0, s_0)) = (\mathbf{r}_3, s_3)$. Any query that has to compute patterns or models is called an *inductive query*. Notice that when the user needs the evaluation functions (e.g., frequencies), their values are computed from the current data part of the inductive database instance.
This approach is studied in depth in the cInQ consortium[2] and has lead to interesting results for local pattern discovery (item sets, association rules, linear graphs, sequences, strings, see [4, 10] for survey papers and an introduction to the terminology). In this paper, we consider a pattern domain related to item sets since biologists consider that useful knowledge about the transcriptome can be expressed as sets of genes and/or sets of biological situations that have some properties. As a methodological guideline, specifying a pattern domain leads to the definition of pattern languages, evaluation functions and primitive constraints. A query language must enable to define standard queries on the data component but also inductive queries. As a first approximation, we can consider that inductive queries (i.e., selection predicates) are built from boolean combinations of primitive constraints.

Let us now consider the languages we have for the `RNA` pattern domain. Let $\mathcal{S}$ denote a set of biological situations and $\mathcal{A}$ denote a set of genes. An expression matrix (ED) associates each couple of $\mathcal{S} \times \mathcal{A}$ a real number, i.e., an expression value. It is out of the scope of this paper to discuss its semantics (exact number of sequenced tags[3] - or absolute frequency- in the case of SAGE data [23], variation between two studied experimental conditions in the case of microarray data [12]).
Raw expression data can be stored in, e.g., relational databases, and be queried by any standard query language (selection of situations, projection on sets of genes). Also, aggregates can be used to derive summarization of raw data, e.g., the expression mean value or the standard variation for each gene. It makes sense to abstract a raw expression matrix into a

[1] Selection of data and patterns are respectively denoted by σ and τ. As it is clear from the context, the operation can also be applied on inductive database instances.
[2] European Contract IST-2000-26469, **c**onsortium on discovering knowledge using **In**ductive **Q**ueries.
[3] For SAGE, tags correspond to genes and the biological situations are called libraries.

boolean matrix $\mathbf{r}$ that records expression properties. In the example from Figure 1, $\mathcal{S} = \{s_1, \ldots s_5\}$ and $\mathcal{A} = \{a_1, a_2, \ldots a_{10}\}$. Each attribute a_j denotes a property about the expression of gene j. The expression data is thus represented by the matrix $\mathbf{r}$ of the binary relation $R \subset \mathcal{S} \times \mathcal{A}$ defined for each situation and each attribute. $(s_i, a_j) \in \mathbf{r}$ denotes that situation i has the property j, e.g., that gene j is over-expressed (under-expressed, has a strong variation) in situation i. In the following, we assume that expression properties encode over-expressions.

The boolean data to be mined is thus a 3-tuple $\mathbf{r} = (\mathcal{S}, \mathcal{A}, R)$.

The `RNA` pattern language is the collection of couples from $\mathcal{L}_\mathcal{A} \times \mathcal{L}_\mathcal{S}$ where $\mathcal{L}_\mathcal{A} = 2^\mathcal{A}$ (sets of genes) and $\mathcal{L}_\mathcal{S} = 2^\mathcal{S}$ (sets of situations). For instance, a typically interesting pattern for a biologist can be (X, T) where X is a set of genes that include at least one transcription factor and which are consistently up-regulated in all the normal lung tissues, i.e., set T.

	Attributes									
Situations	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}
s_1	1	1	1	1	0	1	1	0	0	0
s_2	1	1	1	1	0	0	0	0	1	1
s_3	1	1	1	1	0	0	0	0	1	1
s_4	0	0	0	0	1	1	1	1	1	1
s_5	1	0	1	0	1	1	1	1	0	0

Fig. 1. Example of a boolean matrix $\mathbf{r}_1$

To obtain boolean matrices from a raw expression matrix ED, we use one discretization operator. Typically, such an operator assigns the true value when the expression value is above some threshold value. Let us denote by β_1 and β_2 two operators, $\beta_1(ED) = \mathbf{r}_1$ and $\beta_2(ED) = \mathbf{r}_2$ are two boolean matrices that are generally different. Important properties like containment can be proved or studied (see Section 4 for examples).

Among the simple boolean data transformations, *transposition* can be used. If $\mathbf{r} = (\mathcal{S}, \mathcal{A}, R)$ is a boolean expression matrix, the transposed matrix is ${}^t\mathbf{r} = (\mathcal{A}, \mathcal{S}, {}^tR)$ where $(a, s) \in {}^tR \iff (s, a) \in R$.

Let us now consider evaluation functions for the `RNA` pattern domain. We do not claim that this is an exhaustive list of useful primitives on situations $\times$ genes expression matrices. We mainly consider Galois operators (see, e.g., [24]) that have been proved extremely useful.

Definition 1 (Galois connection [24]). *If $T \subseteq \mathcal{S}$ and $X \subseteq \mathcal{A}$, assume $f(T, \mathbf{r}) = \{a \in \mathcal{A} \mid \forall s \in T, (s, a) \in R\}$ and $g(X, \mathbf{r}) = \{s \in \mathcal{S} \mid \forall a \in X, (s, a) \in R\}$. f provides the set of over-expressed genes that are common to a set of situations and g provides the set of situations that share a given set of attributes (expression properties). (f, g) is the so-called Galois connection between $\mathcal{S}$ and $\mathcal{A}$. We use the classical notations $h = f \circ g$ and $h' = g \circ f$ to denote the Galois closure operators.*

Definition 2 (Frequency). *The* frequency *of a set of genes $X \subseteq \mathcal{A}$ denoted $\mathcal{F}(X, \mathbf{r})$ is the size of $g(X, \mathbf{r})$. The frequency of a set of situations $T \subseteq \mathcal{S}$ is the size of $f(T, \mathbf{r})$.*

Given Figure 1 (parameter $\mathbf{r}_1$ is omitted), let us consider the pattern (X, T) where $X = \{a_1, a_3\}$ and $Y = \{s_1, s_2, s_3, s_5\}$. We have $\mathcal{F}(X) = 4$ and $T = g(X)$.

Let us now introduce some primitive constraints for the `RNA` pattern domain. Our primitive constraints are defined on sets of genes and sets of situations.

Definition 3 (Constraints on frequencies). *Given a set of genes $X \subseteq \mathcal{A}$ and an absolute frequency threshold γ, $\mathcal{C}_{\text{minfreq}}(X, \mathbf{r}) \equiv \mathcal{F}(X, \mathbf{r}) \geq \gamma$. Sets that satisfy $\mathcal{C}_{\text{minfreq}}$ are said* γ-frequent *in $\mathbf{r}$. The maximal frequency is defined as $\mathcal{C}_{\text{maxfreq}}(X, \mathbf{r}) \equiv \mathcal{F}(X, \mathbf{r}) \leq \gamma$. These constraints can be defined on sets of situations as well.*

Definition 4 (Closed set and $\mathcal{C}_{Close}$ constraint). *A set of genes $X \subseteq \mathcal{A}$ is closed (it satisfies the $\mathcal{C}_{Close}$ constraint in $\mathbf{r}$) iff $h(X, \mathbf{r}) = X$. A set of situations $T \subseteq \mathcal{S}$ is closed iff $h'(T, \mathbf{r}) = T$.*

Given Figure 1, assume the `RNA` pattern $(\{a_1, a_2\}, \{s_1, s_2, s_3\})$. if $\gamma = 3$, $\{a_1, a_2\}$ satisfies $\mathcal{C}_{\text{minfreq}}$ in $\mathbf{r}_1$. Furthermore, $\{s_1, s_2, s_3\} = g(\{a_1, a_2\})$. $\{s_1, s_2, s_3\}$ is a closed set of situations (i.e., $h'(\{s_1, s_2, s_3\}) = \{s_1, s_2, s_3\}$) but $\{a_1, a_2\}$ is not a closed set on genes: $h(\{a_1, a_2\}) = f(g(\{a_1, a_2\})) = \{a_1, a_2, a_3, a_4\}$.
The closure of a set of genes X, $h(X, \mathbf{r})$, is the maximal (w.r.t. set inclusion) superset of X which has the same frequency than X in $\mathbf{r}$. A closed set of genes is thus a maximal set of genes whose expression properties (true values) are shared by a set of situations. For instance, the closed set $\{a_1, a_3\}$ in $\mathbf{r}_1$ (see Figure 1) is the largest set of genes that are over-expressed simultaneously in situations s_1, s_2, s_3 and s_5. The Galois connection gives rise to *concepts* [24] that associate closed sets of genes with closed sets of situations.

Definition 5 (Concept). *If $X \in \mathcal{L}_{\mathcal{A}}$ and $T \in \mathcal{L}_{\mathcal{S}}$, we say that (X, T) is a concept in $\mathbf{r}$ when $T = g(X, \mathbf{r})$ and $X = f(T, \mathbf{r})$. By construction, concepts are built on closed sets and each closed set of genes (resp. situations) is linked to a closed set of situations (resp. genes) [24].*

Seven `RNA` patterns are concepts in $\mathbf{r}_1$ (see Figure 1). Examples of concepts are $(\{a_1, a_3\}, \{s_1, s_2, s_3, s_5\})$ and $(\{a_1, a_2, a_3, a_4, a_9, a_{10}\}, \{s_2, s_3\})$. $(\{a_1, a_2\}, \{s_1, s_2, s_3\})$ is not a concept.
Concept are interesting for the biologist: they can suggest the so-called transcription modules.

An important kind of primitive constraint concerns the *syntactical restrictions* that can be checked without any access to the data. [17] contains a systematic study of syntactical constraints for sets.

Definition 6 (Syntactic constraints). *A syntactic constraint enforces that a set $Y \in \mathcal{L}_C$, where $\mathcal{L}_C \subseteq \mathcal{L}_{\mathcal{A}}$ or $\mathcal{L}_C \subseteq \mathcal{L}_{\mathcal{S}}$. Various means can be used to specify $\mathcal{L}_C$, e.g., regular expressions.*

Some other interesting constraints can use additional information about the genes or the situations. For instance, given a set of genes X, it is possible to use biological knowledge about gene functions and enforce constraints on gene functions for the genes in X.

Many data mining processes on gene expression matrices can be formalized as the computation of `RNA` patterns whose set components satisfy combinations of primitive constraints.

Mining the frequent sets of genes is specified as the computation of $\{X \in \mathcal{L}_{\mathcal{A}} \mid \mathcal{C}_{\text{minfreq}}(X, \mathbf{r})$ satisfied$\}$. We can then provide each `RNA` pattern of the form $(X, g(X, \mathbf{r}))$ where X is frequent. This collection can suggest synexpression groups. Adding syntactical constraints (e.g., enforcing the presence or the absence of some genes) is also often used by biologists. Frequent sets of situations can be desired as well.

Mining the closed sets of genes is specified as the computation of $\{X \in \mathcal{L}_{\mathcal{A}} \mid \mathcal{C}_{Close}(X, \mathbf{r})$ satisfied$\}$. These sets provide a valuable information to biologists thanks to closeness property, e.g., closed sets of genes are maximal sets of genes that are co-regulated in a set of situations. In that context, each `RNA` pattern of the form $(X, g(X, \mathbf{r}))$ is a concept. Dually, it is possible to compute closed sets on situations T and associate the closed set of genes $f(T, \mathbf{r})$. A typically useful task is to look for every concept (X, T) such that X is frequent. Syntactical restrictions can be used as well.
Many other examples could be given, e.g., feature construction by looking for sets of genes that satisfy $\mathcal{C}_{\text{minfreq}}$ in one data set and $\mathcal{C}_{\text{maxfreq}}$ in another one [11]. As a typical interesting finding for a biologist, one might look for each `RNA` pattern (X, T) such that X is a set of genes that are frequently up-regulated in all the medulloblastomas ($T = g(X)$) and that are infrequently found up-regulated in corresponding normal regions of the brain.
Post-processing queries can be understood as queries on materialized collections of sets: the user selects the sets that fulfill some new criteria. Notice however that, from the specification point of view, they are not different from data mining queries even though the evaluation does not need an extraction phase.

3 Inductive query evaluation

In this section, our goal is to emphasize that the current algorithmic know-how can tackle the evaluation of the kind of inductive query we need thanks to a clever use of the Galois connection. We chose to emphasize constraint-based extraction of sets of genes for which it is then possible to associate sets of situations (using the g operator).
Mining frequent sets has been extensively studied the last 10 years. One major recent progress comes from the various algorithms that compute efficiently the sets that satisfy a conjunction of anti-monotonic and monotonic constraints, e.g., [17, 11, 15, 7]. Indeed, most of the primitive constraints we have considered are monotonic or anti-monotonic. Some of them, the succinct ones [17], are in fact syntactical constraints

that can be put transformed into a conjunction of monotonic and anti-monotonic constraints. We assume the reader knows well the background in constraint-based mining and the efficient use of monotonicity.
Expression matrices have generally a few tens of lines (biological situations) and thousands or even tens of thousands of columns (genes). Thus, the computation of sets of genes that satisfy a given constraint $\mathcal{C}$ is extremely hard. Indeed, as soon as we have more than a few tens of columns, only a quite small subset of the search space can be explored. Then, the size of the solution, i.e., the collection of the sets that satisfy $\mathcal{C}$ can be so huge that no algorithm can compute them all. When a constraint like $\mathcal{C}_{\text{minfreq}}$ is used, it is possible to take a greater frequency threshold to decrease a priori the size of the solution. The used threshold can however be disappointing for the biologist: extracted patterns are so frequent that they are already known (e.g., they are the so-called house-keeping genes). Furthermore, in the expression matrices we have to analyze, the number of the frequent sets can be huge, whatever is the frequency threshold. It comes from the rather low number of lines and thus the small number of possible frequencies. Clearly, Apriori-like algorithms that have to compute the frequency of at least every frequent set can not be used here. Any Apriori-based strategy for pushing the other constraints might fail too.
When the minimal frequency constraint is used, one of the key idea for inductive query optimization in `RNA` can come from the condensed representation of the frequent sets. They contain a much smaller number of sets with their frequencies even though it is straightforward and efficient to regenerate all the frequent sets and their frequencies. Various condensed representations have been studied, see, e.g., [8, 9]. Indeed, it is easy to derive the whole collection of the frequent sets of genes from $\{X \in \mathcal{L}_{\mathcal{A}} \mid \mathcal{C}_{\text{minfreq}}(X, \mathbf{r}) \wedge \mathcal{C}_{Close}(X, \mathbf{r}) \text{ satisfied}\}$. This compact representation can be computed efficiently, see, e.g., [20, 5, 21, 25, 1].
The algorithm we use is based on free set extraction [5]. Freeness characterizes the closed set generators (i.e., the closures of the free sets are the closed sets).

Definition 7 (Freeness and $\mathcal{C}_{\text{free}}$ constraint). *A set of genes $X \subseteq \mathcal{A}$ is free iff the frequency of X in $\mathbf{r}$ is strictly lower than the frequency of every strict subset of X. We say that X satisfies constraint $\mathcal{C}_{\text{free}}$ in $\mathbf{r}$. Interestingly, freeness is an anti-monotonic property while closeness is not an anti-monotonic one.*

Given Figure 1, $\{a_1, a_6\}$ satisfies $\mathcal{C}_{\text{free}}$ in $\mathbf{r}_1$ but $\{a_1, a_2, a_3\}$ does not.

In other terms, we compute the collection of the closed sets of genes as $\{h(X, \mathbf{r}) \in \mathcal{L}_{\mathcal{A}} \mid \mathcal{C}_{\text{free}}(X, \mathbf{r}) \text{ satisfied}\}$. Minimal frequency constraint can be added as well.
Even though these approaches have given excellent results on large matrices for transactional data (e.g., highly correlated and rather dense data in WWW usage mining applications), they often fail on expression matrices because of the their "pathological" dimensions. Furthermore, we want to enable the use of various discretization operators and thus the analysis of more or less dense matrices. It appeared crucial to us that we can achieve a breakthrough w.r.t. extraction feasibility.

We have studied the extraction from the transposed matrices using the Galois connection to infer the results that would have been extracted from the initial matrices. [22] provides a general framework for transposed extractions given a constraint $\mathcal{C}_{\text{minfreq}}$ with the frequency threshold greater than 1. In a context where the number of columns is quite large w.r.t. the number of lines, i.e., the case for gene expression matrices, it is possible to compute every concept (absolute frequency threshold set to 1) based on the following observation:

- The direct extraction computes the closed sets of genes and for each closed set X ($h(X, \mathbf{r}) = X$), computing $g(X, \mathbf{r}) = T$ enables to provide the concept (X, T). This computation can be intractable due to the number of genes.
- The transposed extraction computes the closed sets of situations and for each closed set T ($h(T, {}^t\mathbf{r}) = h'(T, \mathbf{r}) = T$), computing $g(T, {}^t\mathbf{r}) = f(T, \mathbf{r}) = X$ enables to provide the concept (X, T).
- Computing $g(X, \mathbf{r})$ during the direct extraction or $g(T, {}^t\mathbf{r})$ during the transposed extraction can be performed at almost no cost during the computations of the associated free sets and their closures.

Thus, it is possible to obtain the same collection of concepts when extracting them from a matrix or its transposed. The choice between one or the other method can be guided by the dimension of the matrix: for expression matrices, our experience is that transposition is often needed. It is important to know that when concepts are obtained, it is straightforward to provide frequent sets (for genes and situations) and more generally, many constraint-based extractions of `RNA` patterns can be performed by filtering techniques and, eventually, partial regeneration phases.

4 Applications to SAGE data

We are working with the publicly available SAGE data produced from human cells[4] and it leads to difficult data mining contexts.
Analyzing human SAGE data is relevant since this data source has been largely under-exploited today. The only available on line approach consists in comparing the existing libraries 2 by 2 to extract differential information. To the best of our knowledge, [18] is the unique study on the complete human SAGE data mining. One obvious reason for such a poor exploitation lies in the structure of the data, including a high error rate for low frequency tags (and especially tags appearing only once in a library). The use of discretization operators provides a solution to the problem of low frequency tags. It is our conviction that some essential biological information might be derived from the mass of the SAGE data. We designed a relational database for storing the data available on the NCBI site. From such a database, we have built two gene expression matrices, the so-called 74×822 and 90×12636 matrices.
The construction of the 74×822 matrix is described in [2]. It records the expression level, as of June 2001, for 822 genes belonging to the minimal transcriptome set [23]. After having extracted biologically relevant information from the 74×822 matrix for which the direct extraction worked

[4] www.ncbi.nlm.nih.gov/SAGE/index.cgi

quite well, we decided to build the most exhaustive SAGE expression matrix,i.e., to the best of our knowledge, an original contribution to human SAGE data analysis. From the human libraries available in December 2002. We selected those with more than 20 000 tag sequences and we eliminated the tag sequences for which the identification was ambiguous, based on the SAGE map file [5]. Numerous tags are present only once in a library. Nowadays, it is difficult to evaluate whether these tags represent some real genes or correspond to sequencing errors. We therefore only kept the tags appearing at least twice in at least one library. It as provided a matrix of 90 libraries and 27 679 tags. Using simple statistics (see [3]), we have been able to remove 15 043 tags. In the end, we produced a matrix recording the expression level of 90 libraries and 12 636 tags.

We have chosen to extract the information concerning the over expression of genes: the true value (1) for one given library and one given gene indicates the over expression of this gene in this library. On the contrary, a false value (0) indicates that this gene is not over expressed in this library. We have studied four discretization techniques:

- "ENE" for "Expressed or not". We assign the value 1 when the tag is present (whatever is its value) in the library, 0 otherwise.
- "Mid-Ranged". The highest and lowest expression values in the library are identified for each tag and the mid-range value is defined as being equidistant from these two numbers (arithmetic mean). Then, for a given tag, all expression values that are strictly above the mid-range value give rise to value 1, 0 otherwise.
- "Max - X% Max". The cut off is fixed w.r.t. the maximal expression value observed for each tag. From this value, we deduce a percentage X of this value, 25% in our experiments to decide for over expression.
- "X% Max". For each tag, we consider libraries in which its level of expression is in X% of the highest values (5% in our experiments). These tags are assigned to value 1, 0 for the others.

These different discretization procedures give rise to boolean matrices with varying densities (number of true values on the number of values, see Columns 2 in Table 1). It estimates the difficulty of the extractions. From a qualitative point of view, there is no good discretization method. The impact of the discretization on the validity/interestingness of the extracted regularities must be studied in each particular context. A typical mining task is then to look at $\tau_C(\beta_i(ED)) \cap \tau_C(\beta_j(ED))$, i.e., looking at the similar sets of genes that have been found by two different discretization operators. Indeed, many useful tasks will also involve other set operations between the pattern collections.

We have used the `mv-miner` prototype developed by F. Rioult with an absolute frequency threshold of 1. In that context, it provides each free set on the columns, its frequency, its closure (i.e., a closed set on the columns) and its associated closed sets w.r.t. the lines (Pentium 800MHz with RAM 4GB and 3GB for swap, linux operating system). We have compared the extraction performances not only between the large and the smaller matrices but also across the different discretizations. The

[5] ftp://ftp.ncbi.nih.gov/pub/sage/map/Hs/NlaIII/

	Discretization	Density	Nb free sets	Nb closed sets
M1	ENE	82.8	intractable	intractable
^{t}M1	ENE	82.8	intractable	intractable
M1	Mid-Ranged	12.2	13 580 544	80 068
^{t}M1	Mid-Ranged	12.2	209 829	80 068
M1	Max - 25% Max	3.8	35 934	1 386
^{t}M1	Max - 25% Max	3.8	3 211	1 386
M1	5% Max	4.8	72 630	1 808
^{t}M1	5% Max	4.8	3 362	1 808

	Discretization	Density	Nb free sets	Nb closed sets
M2	ENE	34.5	intractable	intractable
^{t}M2	ENE	34.5	intractable	intractable
M2	Mid-Ranged	4.8	intractable	intractable
^{t}M2	Mid-Ranged	4.8	324 565	196 130
M2	Max - 25% Max	2.2	intractable	intractable
^{t}M2	Max - 25% Max	2.2	21 603	9 150
M2	5% Max	4.7	intractable	intractable
^{t}M2	5% Max	4.7	54 762	31 766

Table 1. Results for $M1 = 74 \times 822$ and $M2 = 90 \times 12\ 636$

results on the boolean matrices derived from 74×822 (resp. $90 \times 12\ 636$) are in Table 1.
In these contexts, one database scan does not cost too much and extraction time is clearly related to the number of generated candidates. Only the numbers of free sets (with frequency $\neq 0$) and closed sets in both a boolean matrix and its transposed matrix are given here. Results are very interesting. Intractability in very high density matrices is understandable. In every case for the larger matrices, extraction becomes feasible by working on the transposed matrix. For instance, using the Mid-Ranged discretization on the large $90 \times 12\ 636$ matrix, the process has failed after more than 17 hours of computations while it has taken less than 1 minute on its transposed version. Others preliminary experiments on microarray data [22] have confirmed the added-value of the approach.

5 Conclusion

We are studying an inductive database approach to gene expression data analysis. Among others, it enforces us to think in terms of primitive constraints and efficient constraint-based mining techniques. Efficiency is needed not only for tractability but also for supporting the dynamic aspects of knowledge discovery (interactivity with the biologists).
We have identified a small set of primitives that are quite useful for real gene expression data analysis. Expressing analysis process by means of a sequence of queries is also important for optimizing the computations

when, e.g., new expression data is available (a new evaluation of a potentially complex process from a new initial expression matrix).
Surprisingly, the pathological dimensions of the expression matrices that were for us a bottleneck a few months ago, enable now to extract every concept in real data. It comes from the combination of an efficient algorithm for computing the closed sets from the free sets and the use of the Galois connection properties.
We are currently carrying the experimentation on the large SAGE matrix to extract biologically meaningful groups of co-regulated genes and their associated sets of biological situations. This should result in more biologically interesting findings than in [2] since the large matrix records the expression level of far more genes, and therefore of far more particular genes of special interest to a given biologist.

Acknowledgements. This work has been partially funded by the EU contract cInQ IST-2000-26469 (FET arm of the IST programme) and a French inter-EPST Bioinformatic program (2002-2003) through which Céline Robardet is supported. François Rioult is supported by the IRM department of the Caen CHU, the comité de la Manche de la Ligue contre le Cancer and the Conseil Régional de Basse-Normandie. Sylvain Blachon is supported by the comité de Saône et Loire de la Ligue contre le Cancer.

References

1. Y. Bastide, R. Taouil, N. Pasquier, G. Stumme, and L. Lakhal. Mining frequent patterns with counting inference. *SIGKDD Explorations*, 2(2):66 – 75, Dec. 2000.
2. C. Becquet, S. Blachon, B. Jeudy, J.-F. Boulicaut, and O. Gandrillon. Strong association rule mining for large gene expression data analysis: a case study on human SAGE data. *Genome Biology*, 12, 2002.
3. S. Blachon, C. Robardet, J-F. Boulicaut, and O. Gandrillon. Extraction de régularités dans des données d'expression SAGE humaines. In *Proceedings Informatique et analyse du transcriptome JPGD'03*, Lyon, F, May 2003. In French.
4. J.-F. Boulicaut. Inductive databases and multiple uses of frequent itemsets: the cInQ approach. In *Database Support for Data Mining Application*, Rosa Meo et al. Eds. Springer-Verlag LNCS 2682, In Press. To appear.
5. J.-F. Boulicaut, A. Bykowski, and C. Rigotti. Approximation of frequency queries by mean of free-sets. In *Proceedings PKDD'00*, volume 1910 of *LNAI*, pages 75–85, Lyon, F, Sept. 2000. Springer-Verlag.
6. J.-F. Boulicaut, M. Klemettinen, and H. Mannila. Modeling KDD processes within the inductive database framework. In *Proceedings DaWaK'99*, volume 1676 of *LNCS*, pages 293–302, Florence, I, Sept. 1999. Springer-Verlag.
7. C. Bucila, J. Gehrke, D. Kifer, and W. White. DualMiner: A dual pruning algorithm for itemsets with constraints. *Data Mining and Knowledge Discovery* 7(3):241–272, 2003.

8. A. Bykowski. *Condensed representations of frequent sets: application to descriptive pattern discovery.* PhD thesis, INSA Lyon, F-69621 Villeurbanne cedex, France, Oct. 2002.
9. T. Calders and B. Goethals. Mining all non derivable frequent itemsets. In *Proceedings PKDD'02*, volume 2431 of *LNAI*, pages 74–83, Helsinki, FIN, Aug. 2002. Springer-Verlag.
10. L. De Raedt. A perspective on inductive databases. *SIGKDD Explorations*, 4(2):69–77, January 2003.
11. L. De Raedt and S. Kramer. The levelwise version space algorithm and its application to molecular fragment finding. In *Proceedings IJCAI'01*, pages 853 – 862, Seattle, USA, Aug. 2001. Morgan Kaufmann.
12. J. L. DeRisi, V. R. Iyer, and P. O. Brown. Exploring the metabolic and genetic control of gene expression on a genomic scale. *Science*, 278, 1997.
13. M. Eisen, P. Spellman, P. Brown, and D. Botstein. Cluster analysis and display of genome-wide expression patterns. *Proceedings National Academy of Science USA*, 95:14863–14868, 1998.
14. T. Imielinski and H. Mannila. A database perspective on knowledge discovery. *CACM*, 39(11):58–64, Nov. 1996.
15. B. Jeudy and J.-F. Boulicaut. Optimization of association rule mining queries. *Intelligent Data Analysis*, 6(4):341 – 357, 2002.
16. H. Mannila and H. Toivonen. Levelwise search and borders of theories in knowledge discovery. *Data Mining and Knowledge Discovery journal*, 1(3):241–258, 1997.
17. R. Ng, L. V. Lakshmanan, J. Han, and A. Pang. Exploratory mining and pruning optimizations of constrained associations rules. In *Proceedings of ACM SIGMOD'98*, pages 13–24, Seattle, USA, May 1998. ACM Press.
18. R. Ng, J. Sander, and M. Sleumer. Hierarchical cluster analysis of sage data for cancer profiling. In *Proceedings BIOKDD'01 co-located with ACM SIGKDD'01*, San Francisco, USA, Aug. 2001.
19. C. Niehrs and N. Pollet. Synexpression groups in eukaryotes. *Nature*, 402:483–487, 1999.
20. N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Efficient mining of association rules using closed itemset lattices. *Information Systems*, 24(1):25–46, Jan. 1999.
21. J. Pei, J. Han, and R. Mao. CLOSET an efficient algorithm for mining frequent closed itemsets. In *Proceedings SIGMOD Workshop DMKD'00*, Dallas, USA, May 2000.
22. F. Rioult, J.-F. Boulicaut, B. Crémilleux, and J. Besson. Using transposition for pattern discovery from microarray data. In *Proceedings SIGMOD Workshop DMKD'03*, pages 73–79, San Diego, USA, June 2003.
23. V. Velculescu, L. Zhang, B. Vogelstein, and K. Kinzler. Serial analysis of gene expression. *Science*, 270:484–487, 1995.
24. R. Wille. Restructuring lattice theory: an approach based on hierarchies of concepts. In *Ordered sets*, pages 445–470. Reidel, 1982.
25. M. J. Zaki. Generating non-redundant association rules. In *Proceedings SIGKDD'00*, pages 34 – 43, Boston, USA, Aug. 2000. ACM Press.

Generalized Version Space Trees

Ulrich Rückert and Stefan Kramer

Technische Universität München
Institut für Informatik/I12
Boltzmannstr. 3
D-85748 Garching b. München, Germany {rueckert, kramer}@in.tum.de

Abstract. We introduce generalized version space trees, a novel data structure that serves as a condensed representation in inductive databases for graph mining. Generalized version space trees allow for a comfortable representation of version spaces and a natural way to efficiently process inductive queries and operations on version spaces. In particular, we focus on using generalized version space trees in the field of mining free (i.e. unrooted) trees in graph databases. Experiments with two data sets from the National Cancer Institute's Developmental Therapeutics Program (DTP) indicate that generalized version space trees can boost inductive queries considerably while featuring moderately increased space requirements in comparison to a version space representation based on boundary sets.

1 Introduction

In order to make inductive databases acceptable for a wider group of users, they should be able to answer queries fast enough to enable interactive usage. In particular for mining in rich (graph or logic-style) representations, we are currently far from this ideal. This is partly due to the very high branching factor of the search space and the existence of syntactic variants aggravating the problems. In graph mining, database scans are expensive, since they involve NP-hard subgraph isomorphism tests for general graphs, and still costly subtree isomorphism tests for free (that is, unrooted) trees. To reduce database scans, so-called condensed representations have been proposed by several authors [BBR00,GLD02,RK01]. Condensed representations are "data structures that make it possible to answer queries about the inductive database approximately correct and reasonably efficient" [Man97] – without database access. Dominique Laurent [GLD02] defines condensed representations as subsets of the solution patterns that allow their complete reconstruction. A variety of condensed representations has been proposed, e.g., free and closed sets (for itemsets) [BBR00], version spaces [RK01] or combinations thereof [GLD02]. De Raedt *et al.* [RJLM02] introduced a data structure called version space trees, that is used for constructing and representing version spaces of strings. Version space trees are, in essence, suffix tries adapted for a boundary set representation of strings. In this paper, we present *generalized version space trees (GVSTs)*, that extend the original data structure to free trees and connected graphs. In principle there is no restriction to extend this data structure even further, for instance to unconnected graphs.

To keep the paper focused, we concentrate on mining free trees in general graphs. In a companion paper [RK03], we have introduced free trees into graph mining, and

presented a canonical form that can be efficiently used in a gSpan-type [XY02] graph mining algorithm. The intention was to overcome the representational limitations of linear path patterns [RK01], while avoiding the complexity issues with general subgraph patterns [IWNM02]. In this paper, we investigate the use of free trees in constraint-based mining: we consider conjunctive queries consisting of a minimum frequency and a maximum frequency part. The setting can easily be extended to arbitrary conjunctions of anti-monotonic and monotic constraints. Note that generalized version space trees can in principle be used for many different purposes, but our focus here is their usage as a condensed representation to speed up typical frequent substructure data mining queries. Our preferred view is that generalized version space trees are like index structures built in the background of database systems: if the queries meet the assumptions made for the construction of the index structure, query answering can be fast, otherwise it might take a long time.

Another facet of this work is the ubiquitous time/space trade-off: for answering queries, we might just as well store all patterns along with supplementary information in a hash table. Although access time would be fast, the solution would not be space-efficient and collisions would have to be handled. The other extreme would be to recompute everything anew, accessing the database every time. It is clear that this would be memory-wise unproblematic, but computationally very expensive. The use of condensed representations such as generalized version space trees is intended to enable fast access to information about already encountered patterns in main memory, at the cost of some memory overhead.

This paper is organized as follows: the next section introduces free trees and a canonical form that can be used for mining frequent free trees in graph data. In section 3 we present the generalized version space tree data structure and in section 4 we report some results on using GVSTs for frequent free tree mining. We conclude in section 5.

2 Frequent Free Tree Mining

First of all, let us introduce the free tree mining setting more formally. Given a set of vertices $V = \{v_1, v_2, \ldots, v_n\}$ and a set of edges $E \subseteq V \times V$, $G =_{def} (V, E)$ constitutes a *graph*. Given a set of vertex labels L_V and a set of edge labels L_E, a *labeled graph* is a graph that has a vertex label associated with each node v, denoted by $label(v) \in L_V$ and an edge label associated with each edge, denoted by $label(e) \in L_E$. For our purposes, the class of *connected acyclic labeled graphs* is of special interest. Figure 1 (a) and (b) are examples of graphs in this class. Because of its acyclicity, each connected acyclic labeled graph has at least one node which is connected to the rest of the graph by only one edge, that is, a *leaf*. In such a graph one can label the leaves with zero and the other nodes recursively with the maximal label of its neighbors plus one (figure 1). This yields an unordered, unrooted tree-like structure, a so-called *free tree*. It is a well-known fact that every free tree has at most two nodes which minimize the maximal distance to all other nodes in the tree, the so-called *centers*. In figure 1 the centers of the two free trees (a) and (b) are marked as empty nodes (◯).

There are various ways to store a free tree on a computer. For efficient processing, it is a good idea to use a representation that ensures that two equivalent free trees are

always encoded as the same pattern. Such a representation is called a *canonical form*. There is a broad range of possible canonical forms that can be used for free trees. In the following we will present a particular canonical form, whose properties allow for an efficient use in the FreeTreeMiner algorithm [RK03]. The canonical form of a free tree is derived in a two step algorithm. First, we identify the canonical center and use it as a root, therefore building a rooted unordered tree from a free tree. In the second step we order the nodes in the rooted tree to get an ordered rooted tree, that is, the canonical form.

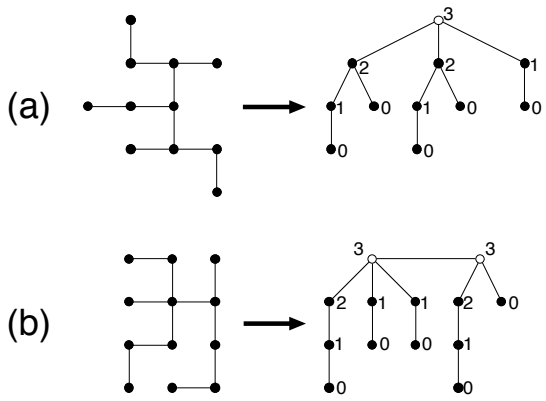


Fig. 1. Illustration of center(s) of free trees. If the leaf nodes of a free tree (i.e. a connected acyclic graph) are labeled with zero and the other nodes are labeled with the maximal label of its neighbors plus one, the nodes with the largest labels are the *centers* of the graph. Every free tree has either one (a) or two (b) centers.

An important building block is a suitable order $\preceq_t$ on ordered rooted trees. In a rooted tree, each edge connects a child node to its parent node. Thus, we can regard each node and the corresponding edge to its parent as a unit. This assigns exactly one label pair $label(e, n) \in L_E \times L_V$ to each node, except for the root node. For the sake of simplicity, we introduce an "empty" label l_ϵ in L_V and L_E. This label is assigned as an replacement for the non-existing parent edge of the root, so that each node n has associated exactly one label $label(e, n) \in L_E \times L_V$. Now, assume we have an arbitrary order $\preceq_{EV}$ on $L_E \times L_V$. In the following we use this order to design a lexicographic order on rooted trees. First we have to introduce some concepts. The *depth* of a node in a tree is the distance to the root node. The set of nodes with the same depth is a *level*. If the tree is ordered, each level can be represented by a sequence of nodes, the *level-sequence*. The *levelwise traversal* of a tree t enumerates the nodes in a tree from the top level to the bottom level of the trees, processing each level-sequence from left to right. If we have two nodes n_1 and n_2 in two ordered trees t_1 and t_2, the *structural completion* of the node n_1 with regard to n_2 is the addition of empty-labeled children to n_1 until

Algorithm 1 An algorithm calculating the levelwise traversal order on two ordered trees t_1 and t_2.

procedure IsSmallerOrEqual(t_1, t_2)
  Append the (root(t_1), root(t_2)) to $queue$
  while $queue$ not empty **do**
    Remove (n_1, n_2) from the front of $queue$
    if label(n_1) $\preceq_{EV}$ label(n_2) **then**
      return true
    else if label(n_2) $\preceq_{EV}$ label(n_1) **then**
      return false
    end if
    for i = 1 to max(| children(n_1) | , | children(n_2) |) **do**
      $c_i \leftarrow \begin{cases} i\text{th child of } n_1 \text{ if } i \leq |\text{ children}(n_1)\ | \\ \text{empty node otherwise} \end{cases}$
      $d_i \leftarrow \begin{cases} i\text{th child of } n_2 \text{ if } i \leq |\text{ children}(n_2)\ | \\ \text{empty node otherwise} \end{cases}$
      Insert (c_i, d_i) at the back of $queue$
    end for
  end while
  return true
end procedure

n_1 has the same number of children than n_2. For example, if n_2 has three children and n_1 has one child, the structural completion adds two children to n_1, both labeled (l_ϵ, l_ϵ). If n_2 has less or equally many children than n_1, the structural completion does not change n_1. The structural completion of the tree t_1 with regard to t_2 is the recursive application of the structural completion for nodes on the nodes of t_1. Intuitively, the structural completion for trees adds empty nodes to t_1 so that each node in t_1 has its unique counterpart in t_2. Finally, the *levelwise traversal order* compares the structural completion of t_1 with regard to t_2 with the structural completion of t_2 with regard to t_1 using the levelwise traversal. Algorithm 1 sketches the idea. A nice property of this order is, that the order of trees with the same size does not change, if we add new nodes at the bottom level nodes. We make use of this property in the FreeTreeMiner algorithm [RK03]. One can also use the levelwise traversal order to order the subtrees in a tree, thus transforming an unordered tree in an ordered tree.

Using this order we can now describe the two steps to build the canonical form of a free tree t. First of all, we identify the centers of t. If there is only one center, we have a unique root. If there are two centers, we remove the edge between the two centers, thus creating two subtrees of the same height. We order the two subtrees and compare them according to the levelwise traversal order. The root of the smaller of the two subtrees is used as the root of the whole tree. Now that we have a rooted tree, we can simply order the nodes in the tree to get the unique canonical representation. Calculating this canonical form can be done in polynomial time.

The canonical form is used in the FreeTreeMiner algorithm outlined in [RK03]. FreeTreeMiner is able to find free trees with a given minimum or maximum support

in graph databases. It builds on the observation that, when working on graph data and using subgraphs as patterns, the canonization task and the subsumption and coverage tests are computationally very expensive: subsumption and coverage tests boil down to subgraph isomorphism, which is NP-complete, and canonization is equivalent to solving the graph isomorphism problem, for which no polynomial algorithm is known, but which has not been proven to be NP-complete either. It is assumed to lie between P and NP. Since FreeTreeMiner uses free trees, it is computationally in a better position: both, subsumption and canonization, can be done in polynomial time.

3 Generalized Version Space Trees

A *generalized version space tree* stores information about a version space in a *suffix trie* structure. A suffix trie is a tree data structure for efficient storage of large numbers of similar strings: its main idea is to decompose each string into substrings and to store one substring per node. The original string can then be reconstructed by following the path from the root to a node and concatenating the substrings in the nodes on the path. De Raedt et al. [RJLM02] show how such a suffix trie, the so called *version space tree*, can be used to represent a version space of strings. In the following we present a suffix trie for version spaces of more general structure, such as free trees and connected graphs. Each node of such a *generalized version space tree* (GVST) represents the canonical form of a free tree (or a connected graph). A free tree that is not in canonical form needs to be transformed to a canonical form prior to storing it in the generalized version space tree. In the following we start with the description of a generalized version space tree for free trees and then show how this structure can be extended to store graphs. The main differences between generalized version space trees and the version space tree outlined in [RJLM02] are

- The version space tree stores only strings, while the generalized version space tree can store more complex structures such as graphs and free trees.
- The version space tree was designed to be used as a data structure for the VST algorithm. Consequently, its nodes contain only information relevant to the query that VST processes. In contrast, the generalized version space tree is designed to be a flexible data structure that can save any number of flags, frequency counts, and utility information in its nodes. It is therefore suitable to support a broad range of queries, including queries containing constraints that are neither anti-monotonic nor monotonic.
- A node in the version space tree contains links to all parents on the previous level. This is feasible for strings and necessary for the VST algorithm, but too costly for complex structures such as free trees. Consequently, generalized version space trees do not link to all parents. This is not a substantial deficit, because many typical operations on generalized version space trees can be processed using simple tree traversal, so that parent links are not needed. Besides it is not required by our gSpan-like algorithm for mining free trees [RK03].

Both, generalized version space trees and string version space trees, allow for efficient implementation of typical operations on version spaces such as calculating the union or

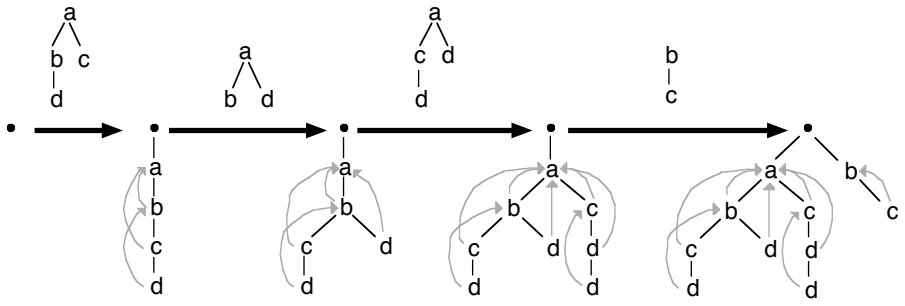


Fig. 2. This figure illustrates the stepwise creation of a generalized version space tree (bottom) from four free trees (top). The black lines denote the parent-child relation, the grey arrows in the generalized version space tree denote the upward "edge parent" link. This link specifies to which parent a new child should be added when reconstructing a free tree from a v-node.

intersection of two version spaces or deciding whether a particular structure is contained in a version space. Typically, those operations can be made by a simple tree traversal and are therefore linear in the number of structures in the tree.

To allow for a clear and concise description, we will denote a node in the general version space tree as a *v-node* and a node in a free tree (that may or may not be contained in the general version space tree) as a *t-node*. Accordingly we will distinguish between *v-edges* and *t-edges*. As explained below, each v-node of a general version space tree contains a t-edge/t-node pair. Such as with suffix tries for strings, the canonical form of a free tree is reconstructed from the general version space tree by following the path from the root v-node to a particular v-node and putting together the t-edge/t-node pairs that are encountered on that path. We denote the free tree that was generated by following the path from the root to the v-node v by t_v. We assume that there is an order $\preceq$ on t-edge/t-node pairs (such as the one outlined in section 2). The children of a v-node are always sorted according to this order. Each v-node contains the following information:

- The labels of a t-edge and a t-node.
- A reference to the parent of this v-node.
- A reference to the leftmost child of this v-node.
- A reference to the right sibling of this v-node.
- The "edge parent" reference to the v-node, that contains the t-node to which this edge-node pair should be appended.
- any number of flags, frequency counts or any derived utility information for the free tree represented by this v-node.

Thus, we store four references and two labels per v-node. The first reference indicates the parent of a node. The next two references are used to link each parent with its children. The first child (according to $\preceq$) can be accessed using the "leftmost child" reference, and all the other children can then be found by repeatedly following the "right sibling" link. Of course, depending on space and performance consideration, one can use any other one-to-many linking scheme to model the parent-children relation, such

Algorithm 2 An algorithm to reconstruct a free tree t_v from a v-node v in a generalized version space tree V.

```
procedure ReconstructTree(v, V)
  path ← path from the root of V to v.
  for n ← first node in path to last node in path do
    if EdgeParent(n) = 0 then
      t_v ← a tree with one node labeled NodeLabel(n)
    else
      Add an edge labeled EdgeLabel(n) and a node labeled NodeLabel(n) to the node in
      t_v that was created during the EdgeParent(n)th iteration
    end if
  end for
  return t_v
end procedure
```

as double-linked lists or hash trees. The fourth reference is more complicated: to create a free tree, one follows the path from the root to a v-node and puts together the t-edges and t-nodes that are encountered on this path. The labels in each v-node specify exactly, which edge and node should be added to the existing free tree, but it does not specify to which t-node the new edge should be connected. This information is given in the fourth reference. Figure 2 illustrates this idea: for the creation of a generalized version space tree, we start with an emtpy generalized version space tree and add a first free tree with four nodes. The t-nodes of the free tree are added to the GVST in levelwise traversal order, i.e. from the top level to the bottom level, on each level from left to right. The "edge parent" reference of a v-node v (denoted by a grey arrow in the figure) identifies the t-parent of the t-node that is represented by v. For example, consider the second generalized version space tree in the figure (i.e. after the insertion of the first free tree): the edge parent of the node labeled `b` is the `a` node, because in the original free tree `b` is a child of `a`. Similarly, the edge parent of `d` is `b`, because in the original free tree `d` is a child of `b`. Thus, when traversing the generalized version space tree from top to bottom, one can easily reconstruct the original free tree by adding a new child to the t-node that corresponds to the v-node of the current v-node's edge parent: the `a` node is created as a root, `b` and `c` are added as children of `a`, and finally `d` is added as child of `b`. This algorithm is outlined in more detail in algorithm 2. Ideally, the edge parent reference is stored as an index into the array containing the v-nodes on the path from the root to the end v-node v of the free tree t_v. The second free tree in figure 2 differs from the first one only on the third node according to the levelwise traversal order. Consequently, the resulting generalized version space tree branches after the second node (labeled `b`). The edge parent reference of a v-node that represents a t-root is set to zero. In our example this is the case for the `a` v-node that represents the roots of the first three trees and the `b` v-node that represents the root of the fourth free tree.

Finding a given free tree t_v in a generalized version space tree V is a straightforward operation: we just climb down the generalized version space tree from the root to the target node v, selecting in each step that child, whose edge and node labels match with the edge and node labels of the "next" edge in t_v. Just as with the insertion operation

we need a way to specify which edge is the "next" edge of a canonical form. This can be expressed by an order $\preceq_t$ on the edges in the free tree. While in principle any consistent order can be used, it makes sense to choose an order that guarantees that the generalized version space tree contains only canonical forms. Such a *canonically inducing* order ensures that for any generalized version space tree V all trees t_w that are constructed on the way from the root node to any node v in V are in canonical form. The levelwise traversal order and the canonical form outlined in section 2 fulfill this criterion only partially: in some seldom cases it might be necessary to include v-nodes in V whose corresponding tree t_v has the wrong canonical center as a root according to the definition of the canonical form. It is quite easy to fix this problem by adding the height of a subtree as another decision criterion to the levelwise traversal order, but this would remove some of the nice properties of the order and hurt FreeTreeMiner's performance. We therefore chose to simply mark any non-canonical v-node in V with a special label and to ignore those non-canonical trees during the find operation. It is easy to see that a generalized version space tree can store any arbitrary rooted labeled tree. If we store only canonical forms, we can therefore build a generalized version space tree containing any arbitrary set of free trees, and in particular version spaces of free trees.

The generalized version space tree can be extended to store connected, labeled, and potentially cyclic graphs as well. The first step to do so is to choose a suitable canonical form and an order on the edges of a graph g. It turns out that Yan and Han's approach in [XY02] works well for our purpose. The main idea is to represent a connected graph g using a *Depth-First Search (DFS) Tree*. Such a tree is generated by following the *forward edges* (i.e. the edges whose end node have not been visited) of a graph in a depth-first style and storing with each visited node a list of *backward edges* (i.e. edges whose end node have already been visited). Of course, there are numerous ways to traverse g in such a depth-first manner depending on the choice of the starting node and the order of the edges for each node in g. Yan and Han combine three orders on forward and backward edges to construct a linear order $\preceq_{E,T}$ on the set of edges E and a linear lexicographic order $\leq$ on DFS trees[1]. One can use this canonical form and the two orders to construct a generalized version space tree V representing connected graphs: again, each v-node v in V identifies the canonical form of exactly one graph g and again one can construct g by following the path from V's root to v and adding iteratively edges to an initially empty graph. The only conceptual difference[2] to the generalized version space tree for free trees is the fact that some of the edges are backward edges, i.e. edges that connect the current node with an already generated node and thus build a cycle. Yan and Han give a proof that the used orders are in fact canonically inducing, i.e. the generalized version space tree contains only canonical forms. Even though Yan and Han lay the groundwork for the design of generalized version space trees, their gSpan algorithm never actually makes use of such a data structure.

[1] In fact, Yan and Han define the order $\leq$ on *DFS-codes*, i.e. sequences that are generated by traversing the graph in a depth-first order constrained by the order $\preceq_{E,T}$ on edges. This is however, just one particular useful way to represent and compare the corresponding DFS trees.

[2] except for the use of different orders

4 Experimental Results

In this section, we present the application of generalized version space trees on two datasets taken from the National Cancer Institute's (NCI) Developmental Therapeutics Program (DTP, http://dtp.nci.nih.gov/).

4.1 NCI DTP Anti-Cancer Screening Data

As a first experiment we examine the performance of generalized version space trees used just like index structures in databases. To this purpose we are applying FreeTreeMiner to the compounds studied in the DTP Human Tumor Cell Line Screen program.

We classified each of the 37,330 compounds in the data set as being "active" or "inactive" depending on the log GI50 value for the NCI-H460 lung cancer line, which can be seen as a measure of the cancer growth inhibition of a compound. This classification separates the compounds in a set of 17,589 "active" compounds and a set of 17,239 "inactive" compounds. We created a list of 100 queries of the form $freq(t, active) \geq p_1 \wedge freq(t, inactive) < p_2$. The values of p_1 and p_2 were chosen for each query randomly from the interval $[0.2; 0.5]$. Note that these support levels are too high for practical applications, but they serve the purpose of our argument here. Finally, we used FreeTreeMiner to calculate the version spaces for all 100 queries. The computation took 24984 seconds, that is on average about 250 seconds per query. In the next step, we use FreeTreeMiner only once to calculate the query $freq(t, active) \geq 0.2 \wedge freq(t, inactive) < 0.5$. The resulting version space is stored in a generalized version space tree. We also save in each node of this tree the frequency counts of the corresponding free tree on both sets. After this preprocessing step we can simply walk through the tree and output all free trees that match a query. The calculation of the generalized version space tree took 399 seconds, that is less than twice the average time needed for computing a query. After the generalized version space tree was computed, all 100 queries were executed in just 0.2 seconds. That means the use of the generalized version space tree as a cache structure reduced the overall processing time from 24984 seconds to 399 seconds, a time saving of over 98%. Clearly, generalized version space trees are an efficient way to boost similar frequent substructure queries on databases. Note that queries of the form $freq(t, D1) \geq p_1 \wedge freq(t, D2) < p_2$ are quite common, because they allow to find patterns whose frequency varies significantly between two given groups of instances.

4.2 NCI DTP Anti-HIV Screening Data

In the second experiment we examine the use of generalized version space trees as a representation for version spaces. Most of the proposed algorithms in the literature simply output all elements in the determined version space. A better approach is to calculate the G- and S-set borders as an efficient and small representation of the version space. Such a representation can then be used during further processing. From a computational point of view, generalized version space trees should be more efficient than borders, because the subsumption tests, that are necessary for the calculation and processing of the borders, are much more time consuming than the lookup operations in a generalized

version space tree. It is, however, not clear, how the space requirements of generalized version space trees interfere with its performance. If a generalized version space tree needs too much space to be kept in main memory, it has to be stored on the much slower external memory such as a hard disk. In such a case it might be more efficient to use a computationally more demanding, but less space consuming representation such as borders. In the following we will examine this question using the DTP HIV data.

The DTP AIDS Antiviral Screen program (http://dtp.nci.nih.gov) has checked tens of thousands of compounds for evidence of anti-HIV activity. Available are screening results and chemical structural data on compounds that are not covered by a confidentiality agreement. The available database (October 1999 Release) contains the screening results for 43,382 compounds. The screen utilizes a soluble formazan assay to measure protection of human CEM cells from HIV-1 infection [WKF+89]. Compounds able to provide at least 50 % protection to the CEM cells were retested. Compounds that provided at least 50 % protection on retest were listed as moderately active (CM, confirmed moderately active). Compounds that reproducibly provided 100 % protection were listed as confirmed active (CA). Compounds neither active nor moderately active were listed as confirmed inactive (CI).

We performed two tests on this data set to gain some insights into the size of generalized version space trees in comparison to a representation using borders. First, we used FreeTreeMiner to calculate the version space containing all free trees with a minimum support of 39 in the CA data set and a maximum support of 10 in the CM data set. The resulting version space includes 834 frequent free trees. The corresponding generalized version space tree uses 15996 nodes. We also calculated the G-set and S-set for this version space. The G-set contains 22 trees, the S-set 255 trees. Together, the trees in both borders use 4162 nodes, or roughly a quarter of the number of nodes in the version space tree. We repeated the experiment with a minimum support of 26 for the CA data set and a maximum support of 10 for the CM data set. This time, the version space tree needed 104749 nodes for a version space tree containing 37875 trees, while the 288 trees in the G-set and the 706 trees in the S-set together took 19011 nodes. Thus, even though borders proved to be a very compact representation in these experiments, the generalized version space tree uses only between four to five times more nodes than the borders. It seems that the increased memory demand of generalized version space trees is a problem only for very large version spaces. At least for the examined application, the version space trees are small enough to be easily stored in the main memory of today's computers. In such a case one is gladly willing to accept a higher memory usage in order to gain rapid access to the free trees and an easy handling of the represented version space.

5 Conclusion

In this paper we have introduced generalized version space trees as a condensed representation that can be used to store version spaces of structured data in an easy and efficient way, and that can be utilized in a database to boost certain types of queries just like an index structure. We show how such GVSTs can be built for free trees and connected graphs and we give some first empirical evidence that hints at the applicability

of GVSTs for substructure pattern mining. Of course, there is a lot of further work that can be done: we are planning to perform an elaborate study comparing GVSTs with boundary set representations for free trees and graphs. Another interesting direction of research deals with the type and handling of queries that can be supported by GVSTs. For example, the nodes in a GVST could also contain flags or data that was generated by evaluating a constraint that is neither monotonic nor anti-monotonic. Finally, one could envisage incorporating GVSTs into inductive databases supported by query languages such as MineRule [MPC96] or MSQL [IV99].

Acknowledgements

We would like to thank the anonymous reviewers for their helpful comments.

References

[BBR00] Jean-Francois Boulicaut, Artur Bykowski, and Christophe Rigotti. Approximation of frequency queries by means of free-sets. In *Proceedings of the Fourth European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD-2000)*, pages 75–85, 2000.

[GLD02] Arnaud Giacometti, Dominique Laurent, and Cheikh Talibouya Diop. Condensed representations for sets of mining queries. In *Proceedings of the First International Workshop on Knowledge Discovery in Inductive Databases (KDID-02)*, 2002.

[IV99] Tomasz Imielienski and Aashu Virmani. Msql: A query language for database mining. *Data Mining and Knowledge Discovery*, 3(4):373–408, 1999.

[IWNM02] Akihiro Inokuchi, Takashi Washio, Yoshio Nishimura, and Hiroshi Motoda. General framework for mining frequent patterns in structures. In *Proceedings of the ICDM-2002 workshop on Active Mining (AM-2002)*, pages 23–30, 2002.

[Man97] Heikki Mannila. Inductive databases and condensed representations for data mining. In *International Logic Programming Symposium*, pages 21–30, 1997.

[MPC96] Rosa Meo, Giuseppe Psaila, and Stefano Ceri. A new SQL-like operator for mining association rules. In T. M. Vijayaraman, Alejandro P. Buchmann, C. Mohan, and Nandlal L. Sarda, editors, *VLDB'96, Proceedings of 22th International Conference on Very Large Data Bases, September 3-6, 1996, Mumbai (Bombay), India*, pages 122–133. Morgan Kaufmann, 1996.

[RJLM02] Luc De Raedt, Manfred Jaeger, Sau Dan Lee, and Heikki Mannila. A theory of inductive query answering. In *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM 2002)*, pages 123–130, 2002.

[RK01] Luc De Raedt and Stefan Kramer. The level-wise version space algorithm and its application to molecular fragment finding. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-01)*, pages 853–862, 2001.

[RK03] Ulrich Rückert and Stefan Kramer. Frequent free tree discovery in graph data, submitted. http://home.in.tum.de/∼rueckert/freetreeminer.pdf, 2003.

[WKF+89] O.S. Weislow, R. Kiser, D.L. Fine, J.P. Bader, R.H. Shoemaker, and M.R. Boyd. New soluble formazan assay for HIV-1 cytopathic effects: application to high flux screening of synthetic and natural products for AIDS antiviral activity. *Journal of the National Cancer Institute*, 81:577–586, 1989.

[XY02] Jiawei Ha Xifeng Yan. gSpan: Graph-based substructure pattern mining. In *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM 2002)*, pages 721–724, 2002.