17th European Conference on Machine Learning (ECML) and
10th European Conference on Principles and Practice of
Knowledge Discovery in Databases (PKDD)

Proceedings of

The 5th International Workshop on Knowledge Discovery in Inductive Databases (KDID'06)

Editors
Sašo Džeroski (Jožef Stefan Institute, Ljubljana, Slovenia)
Jan Struyf (Katholieke Universiteit Leuven, Belgium)

Preface

The 5th International Workshop on Knowledge Discovery in Inductive Databases (KDID 2006) was held on September 18, 2006 in Berlin, Germany, in conjunction with ECML/PKDD 2006: The 17th European Conference on Machine Learning (ECML) and the 10th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD).

Inductive databases (IDBs) represent a database view on data mining and knowledge discovery. IDBs contain not only data, but also generalizations (patterns and models) valid in the data. In an IDB, ordinary queries can be used to access and manipulate data, while inductive queries can be used to generate (mine), manipulate, and apply patterns. In the IDB framework, patterns become "first-class citizens" and KDD becomes an extended querying process in which both the data and the patterns/models that hold in the data are queried. The IDB framework is appealing as a general framework for data mining, because it employs declarative queries instead of ad-hoc procedural constructs. As declarative queries are often formulated using constraints, inductive querying is closely related to constraint-based data mining. The IDB framework is also appealing for data mining applications, as it supports the entire KDD process, i.e., nontrivial multi-step KDD scenarios, rather than just individual data mining operations. The goal of the workshop was to bring together database and data mining researchers interested in the areas of inductive databases, inductive queries, constraint-based data mining, and data mining query languages.

This workshop followed the previous four successful KDID workshops organized in conjunction with ECML/PKDD: KDID'02 held in Helsinki, Finland, KDID'03 held in Cavtat-Dubrovnik, Croatia, KDID'04 held in Pisa, Italy, and KDID'05 held in Porto, Portugal. Its scientific program included nine regular presentations and two short ones, as well as an invited talk by Kiri Wagstaff (Jet Propulsion Laboratory, California Institute of Technology, USA). Invited chapters and extended versions of the selected papers will appear in the post-workshop proceedings, which will be published by Springer in the Lecture Notes in Computer Science (LNCS) series. We wish to thank the invited speaker, all the authors of submitted papers, the program committee members and additional reviewers, and the ECML/PKDD organization committee. KDID 2006 was supported by the European project IQ ("Inductive Queries for Mining Patterns and Models", IST FET FP6-516169, 2005-2008).

August 2006

Sašo Džeroski
Jan Struyf

Organization

Program Chairs

Sašo Džeroski — Jožef Stefan Institute, Dept. of Knowledge Technologies
Jamova 39, 1000 Ljubljana, Slovenia
Saso.Dzeroski@ijs.si
http://www-ai.ijs.si/SasoDzeroski/

Jan Struyf — Katholieke Universiteit Leuven, Dept. of Computer Science
Celestijnenlaan 200A, 3001 Leuven, Belgium
Jan.Struyf@cs.kuleuven.be
http://www.cs.kuleuven.be/~jan/

Program Committee

Hiroki Arimura, Hokkaido University, Japan
Hendrik Blockeel, Katholieke Universiteit Leuven, Belgium
Francesco Bonchi, ISTI-C.N.R., Italy
Jean-François Boulicaut, INSA Lyon, France
Toon Calders, University of Antwerp, Belgium
Luc De Raedt, Albert-Ludwigs-Universität Freiburg, Germany
Minos N. Garofalakis, Intel Research Berkeley, USA
Fosca Giannotti, ISTI-C.N.R., Italy
Bart Goethals, University of Antwerp, Belgium
Jiawei Han, University Illinois at Urbana-Champaign, USA
Ross D. King, University of Wales, Aberystwyth, UK
Giuseppe Manco, ICAR-C.N.R., Italy
Rosa Meo, University of Turin, Italy
Ryszard S. Michalski, George Mason University, USA
Taneli Mielikäinen, University of Helsinki, Finland
Shinichi Morishita, University of Tokyo, Japan
Siegfried Nijssen, Albert-Ludwigs-Universität Freiburg, Germany
Céline Robardet, INSA Lyon, France
Arno Siebes, Utrecht University, The Netherlands
Takashi Washio, Osaka University, Japan
Philip S. Yu, IBM Thomas J. Watson, USA
Mohammed Zaki, Rensselaer Polytechnic Institute, USA
Carlo Zaniolo, UCLA, USA

Additional Reviewers

Emma L. Byrne, Hong Cheng, Francesco Folino, Gemma Garriga, Elio Masciari, Jimeng Sun, Janusz Wojtusiak

Table of Contents

Value, Cost, and Sharing: Open Issues in Constrained Clustering

Kiri L. Wagstaff

Jet Propulsion Laboratory, California Institute of Technology,
Mail Stop 126-347, 4800 Oak Grove Drive, Pasadena CA 91109, USA,
kiri.wagstaff@jpl.nasa.gov

Abstract. Clustering is an important tool for data mining, since it can identify major patterns or trends without any supervision (labeled data). Over the past five years, semi-supervised (constrained) clustering methods have become very popular. These methods began with incorporating pairwise constraints and have developed into more general methods that can learn appropriate distance metrics. However, several important open questions have arisen about which constraints are most useful, how they can be actively acquired, and when and how they should be propagated to neighboring points. This position paper describes these open questions and suggests future directions for constrained clustering research.

1 Introduction

Clustering methods are used to analyze data sets that lack any supervisory information such as data labels. They identify major patterns or trends based on a combination of the assumed cluster structure (e.g., Gaussian distribution) and the observed data distribution. Recently, semi-supervised clustering methods have become very popular because they can also take advantage of supervisory information when it is available. The first work in this area proposed a modified version of COBWEB that enforced pairwise constraints indicating when two items were known *a priori* to either belong to the same cluster (must-link) or different clusters (cannot-link) [1]. It was followed by constrained versions of the k-means and EM clustering algorithms [2, 3]. Later work expanded this approach to accommodate soft constraints (preferences) [4, 5] and to infer new distance metrics over a given data set, based on the available constraints. Some metric learning methods are restricted to accommodating must-link constraints only [6], while others can also accommodate cannot-link constraints [7, 8, 5].

These advances have led to further study of the impact of incorporating constraints into clustering algorithms, particularly when applied to large, real-world data sets. Important issues that have arisen include:

1. Given the recent observation that some constraint sets can *adversely* impact performance, how can we determine the utility of a given constraint set, prior to clustering?

2. How can we minimize the effort required of the user, by active soliciting only the most useful constraints?
3. When and how should constraints be propagated or shared with neighboring points?

This paper contributes descriptions of each of these open questions. In identifying these challenges, and the state of the art in addressing them, we highlight several directions for future research.

2 Open Questions

2.1 Value: How Useful is a Given Set of Constraints?

It is to be expected that some constraint sets will be more useful than others, in terms of the benefit they provide to a given clustering algorithm. For example, if the constraints contain information that the clustering algorithm is able to deduce on its own, then they will not provide any improvement in clustering performance. However, virtually all work to date values constraint sets only in terms of the number of constraints they contain. The ability to more accurately quantify the utility of a given constraint set, prior to clustering, will permit practitioners to decide whether to use a given constraint set, or to choose the best constraint set to use, when several are available.

The need for a constraint set utility measure has become imperative with the recent observation that some constraint sets, even when completely accurate with respect to the evaluation labels, can actually decrease clustering performance [9]. The usual practice when describing the results of constrained clustering experiments is to report the clustering performance averaged over multiple trials, where each trial consists of a set of constraints that is randomly generated from the data labels. While it is generally the case that average performance does increase as more constraints are provided, a closer examination of the individual trials reveals that some, or even many, of them instead cause a drop in accuracy. Table 1 shows the results of 1000 trials, each with a different set of 25 randomly selected constraints, conducted over four UCI data sets [10] using four different k-means-based constrained clustering algorithms. The table reports the fraction of trials in which the performance was lower than the default k-means result, which ranges from 0% up to 87% of the trials.

The average performance numbers obscure this effect because the "good" trials tend to have a larger magnitude change in performance than the "bad" trials do. However, the fact that any of the constraint sets can cause a decrease in performance is unintuitive, and even worrisome, since the constraints are known to be noise-free and should not lead the algorithm astray.

To better understand the reasons for this effect, Davidson et al. [9] defined two constraint set properties and provided a quantitative way to measure them. *Informativeness* is the fraction of information in the constraint set that the algorithm cannot determine on its own. *Coherence* is the amount of agreement between the constraints in the set. Constraint sets with low coherence will be

Table 1. Fraction of 1000 randomly selected 25-constraint sets that caused a drop in accuracy, compared to an unconstrained run with the same centroid initialization (table from Davidson et al. [9]).

	Algorithm			
Data Set	**CKM [2]** Constraint enforcement	**PKM [5]** Constraint enforcement	**MKM [5]** Metric learning	**MPKM [5]** Enforcement and metric learning
Glass	28%	1%	11%	0%
Ionosphere	26%	77%	0%	77%
Iris	29%	19%	36%	36%
Wine	38%	34%	87%	74%

difficult to completely satisfy and can lead the algorithm into unpromising areas of the search space. Both high informativeness and high coherence tend to result in an increase in clustering performance. However, these properties do not fully explain some clustering behavior. For example, a set of just three randomly selected constraints, with high informativeness and coherence, can increase clustering performance on the `iris` data set significantly, while a constraint set with similarly high values for both properties has no effect on the `ionosphere` data set. Additional work must be done to refine these measures or propose additional ones that better characterize the utility of the constraint set.

Two challenges for future progress in this area are: 1) to identify other constraint set properties that correlate with utility for constrained clustering algorithms, and 2) to learn to predict the overall utility of a new constraint set, based on extracted attributes such as these properties. It is likely that the latter will require the combination of several different constraint set properties, instead of being a single quantity, so using machine learning techniques to identify the mapping from properties to utility may be a useful approach.

2.2 Cost: How Can We Make Constraints Cheaper to Acquire?

A single pairwise constraint specifies a relationship between two data points. For a data set with n items, there are $\frac{1}{2}n(n-1)$ possible constraints. Therefore, the number of constraints needed to specify a given percentage of the relationships (say, 10%) increases quadratically with the data set size. For large data sets, the constraint specification effort can become a significant burden.

There are several ways to mitigate the cost of collecting constraints. If constraints are derived from a set of labeled items, we obtain $L(L-1)$ constraints for the cost of labeling only L items. If the constraints arise independently (not from labels), most constrained clustering algorithms can leverage constraint properties such as transitivity and entailment to deduce additional constraints automatically. A more efficient way to obtain the most useful constraints for the least effort is to permit the algorithm to actively solicit only the constraints it needs. Klein et al. [7] suggested an active constraint acquisition method in which a

hierarchical clustering algorithm can identify the m best queries to issue to the oracle. Recent work has also explored constraint acquisition methods for partitional clustering based on a farthest-first traversal scheme [11] or identifying points that are most likely to lie on cluster boundaries [12]. When constraints are derived from data labels, it is also possible to use an unsupervised support vector machine (SVM) to identify "pivot points" that are most useful to label [13].

A natural next step would be to combine methods for active constraint acquisition with methods for quantifying constraint set utility. In an ideal world, we would like to request the constraint(s) which will result in the largest increase in utility for the existing constraint set. Davidson et al. [9] showed that when restricting evaluation to the most coherent constraint sets, the average performance increased for most of the data sets studied. This early result suggests that coherence, and other utility measures, could be used to guide active constraint acquisition.

Challenges in this area are: 1) to incorporate measures of constraint set utility into an active constraint selection heuristic, akin to the MaxMin heuristic for classification [14], so that the best constraint can be identified and queried prior to knowing its designation (must/cannot), and 2) to identify efficient ways to query the user for constraint information at a higher level, such as a cluster description or heuristic rule that can be propagated down to individual items to produce a batch of constraints from a single user statement.

2.3 Sharing: When and How Should Constraints be Propagated to Neighboring Points?

Another way to get the most out of a set of constraints is to determine how they can be propagated to other nearby points. Existing methods that learn distance metrics use the constraints to "warp" the original distance metric to bring must-linked points closer together and to push cannot-linked points farther apart [7, 8, 6, 5]. They implicitly rely on the assumption that it is "safe" to propagate constraints locally, in feature space. For example, if a must be linked to b, and the distance $dist(a, c)$ is small, then when the distance metric is warped to bring a closer to b, it is also likely that the distance $dist(b, c)$ will shrink and the algorithm will cluster b and c together as well. The performance gains that have been achieved when adapting the distance metric to the constraints are a testament to the common reliability of this assumption.

However, the assumption that proximity can be used to propagate constraints is not always a valid one. It is only reasonable if the distance in feature space is consistent with the distances that are implied by the constraint set. This often holds true, since the features that are chosen to describe the data points are consistent with the data labels, which are commonly the source of the constraints. One exception is the `tic-tac-toe` data set from the UCI archive [10]. In this data set, each item is a 3x3 tic-tac-toe board that represents an end state for the game, assuming that the 'x' player played first. The boards are represented with nine features, one for each position on the board, and each one can take on a value of 'x', 'o', or 'b' (for blank). The goal is to separate the boards into two

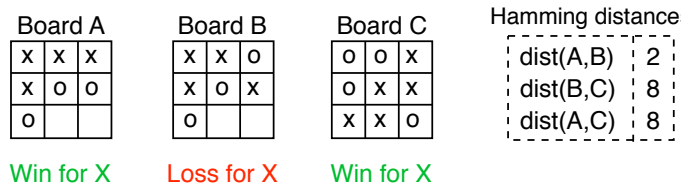


Fig. 1. Three items (endgame boards) from the `tic-tac-toe` data set. For clarity, blanks are represented as blanks, rather than spaces marked 'b'. The Hamming distances between each pair of boards are shown on the right.

clusters: one with boards that show a win for 'x' and one with all other boards (losses and draws).

This data set is challenging because proximity in the feature space does not correlate well with similarity in terms of assigned labels. Consider the examples shown in Figure 1. Hamming distance is used with this data set, since the features have symbolic values. Boards A and B are very similar (Hamming distance of 2), but they should be joined by a cannot-link constraint. In contrast, boards A and C are very different (Hamming distance of of 8), but they should be joined by a must-link constraint. In this situation, propagating constraints to nearby (similar) items will not help improve performance (and may even degrade it).

Clustering performance on this data set is typically poor, unless a large number of constraints are available. The basic k-means algorithm achieves a Rand Index of 51%; COP-KMEANS requires 500 randomly selected constraints to increase performance to 92% [2]. COP-COBWEB is unable to increase its performance above the baseline of 49% performance, regardless of the number of constraints provided [1]. In fact, when we examine performance on a held-out subset of the data[1], it only increases to 55% for COP-KMEANS, far lower than the 92% performance on the rest of the data set. For most data sets, the held-out performance is much higher [2]. The low held-out performance indicates that the algorithm is unable to generalize the constraint information beyond the exact items that participate in constraints. This is a sign that the constraints and the features are not consistent, and that propagating constraints may be dangerous. The results of applying metric learning methods to this data set have not yet been published, probably because the feature values are symbolic rather than real-valued. However, we expect that metric learning would be ineffective in this case.

Challenges to be addressed in this area are: 1) to characterize data sets in terms of whether or not constraints should be propagated (when is it "safe" and when should the data overrule the constraints?), and 2) to determine the degree to which the constraints should be propagated (e.g., how far should the

[1] The data subset is "held-out" in the sense that no constraints were generated on the subset, although it was clustered along with all of the other items once the constraints were introduced.

local neighborhood extend, for each constraint?). It is possible that constraint set coherence [9] could be used to help estimate the relevant neighborhood for each point.

3 Conclusions

This paper outlines several important unanswered questions that relate to the practice of constrained clustering. To use constrained clustering methods effectively, it is important that we have tools for estimating the *value* of a given constraint set prior to clustering. We also seek to minimize the *cost* of acquiring constraints. Finally, we require guidance in determining when and how to *share* or propagate constraints to their local neighborhoods. In addressing each of these subjects, we will make it possible to confidently apply constrained clustering methods to very large data sets in an efficient, principled fashion.

Acknowledgments. I would like to thank Sugato Basu and Ian Davidson for ongoing discussions on constrained clustering issues and their excellent tutorial, "Clustering with Constraints: Theory and Practice," presented at KDD 2006. The research described in this paper was funded by the NSF ITR Program (grant #0325329) and was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

References

1. Wagstaff, K., Cardie, C.: Clustering with instance-level constraints. In: Proceedings of the Seventeenth International Conference on Machine Learning. (2000) 1103–1110
2. Wagstaff, K., Cardie, C., Rogers, S., Schroedl, S.: Constrained k-means clustering with background knowledge. In: Proceedings of the Eighteenth International Conference on Machine Learning. (2001) 577–584
3. Shental, N., Bar-Hillel, A., Hertz, T., Weinshall, D.: Computing Gaussian mixture models with EM using equivalence constraints. In: Advances in Neural Information Processing Systems 16. (2004)
4. Wagstaff, K.L.: Intelligent Clustering with Instance-Level Constraints. PhD thesis, Cornell University (2002)
5. Bilenko, M., Basu, S., Mooney, R.J.: Integrating constraints and metric learning in semi-supervised clustering. In: Proceedings of the Twenty-First International Conference on Machine Learning. (2004) 11–18
6. Bar-Hillel, A., Hertz, T., Shental, N., Weinshall, D.: Learning a Mahalanobis metric from equivalence constraints. Journal of Machine Learning Research **6** (2005) 937–965
7. Klein, D., Kamvar, S.D., Manning, C.D.: From instance-level constraints to space-level constraints: Making the most of prior knowledge in data clustering. In: Proceedings of the Nineteenth International Conference on Machine Learning. (2002) 307–313

8. Xing, E.P., Ng, A.Y., Jordan, M.I., Russell, S.: Distance metric learning, with application to clustering with side-information. In: Advances in Neural Information Processing Systems 15. (2003)
9. Davidson, I., Wagstaff, K.L., Basu, S.: Measuring constraint-set utility for partitional clustering algorithms. In: Proceedings of the Tenth European Conference on Principles and Practice of Knowledge Discovery in Databases. (2006) 115–126
10. Blake, C.L., Merz, C.J.: UCI repository of machine learning databases. http://www.ics.uci.edu/∼mlearn/MLRepository.html (1998)
11. Basu, S., Banerjee, A., Mooney, R.J.: Active semi-supervision for pairwise constrained clustering. In: Proceedings of the SIAM International Conference on Data Mining. (2004) 333–344
12. Xu, Q., desJardins, M., Wagstaff, K.L.: Active constrained clustering by examining spectral eigenvectors. In: Proceedings of the Eighth International Conference on Discovery Science. (2005) 294–307
13. Xu, Q.: Active Querying for Semi-supervised Clustering. PhD thesis, University of Maryland, Baltimore County (2006)
14. Tong, S., Koller, D.: Support vector machine active learning with applications to text classification. Journal of Machine Learning Research **2** (2002) 45–66

Mining bi-sets in numerical data

Jérémy Besson[1,2], Céline Robardet[1], Luc De Raedt[3], and Jean-François Boulicaut[1]

[1] LIRIS UMR 5205 CNRS/INSA Lyon/U. Lyon 1/U. Lyon 2/ECL
INSA Lyon, Bât. Blaise Pascal, F-69621 Villeurbanne, France
[2] UMR INRA/INSERM 1235
F-69372 Lyon cedex 08, France
[3] Albert-Ludwigs-Universitat Freiburg
Georges-Kohler-Allee, Gebaude 079 D-79110 Freiburg, Germany
Contact: celine.robardet@insa-lyon.fr

Abstract. Thanks to an important research effort the last few years, inductive queries on set patterns and complete solvers which can evaluate them on large 0/1 data sets have been proved extremely useful. However, for many application domains, the raw data is numerical (matrices of real numbers whose dimensions denote objects and properties). Therefore, using efficient 0/1 mining techniques needs for tedious Boolean property encoding phases. This is, e.g., the case, when considering microarray data mining and its impact for knowledge discovery in molecular biology. We consider the possibility to mine directly numerical data to extract collections of relevant bi-sets, i.e., couples of associated sets of objects and attributes which satisfy some user-defined constraints. Not only we propose a new pattern domain but also we introduce a complete solver for computing the so-called numerical bi-sets. Preliminary experimental validation is given.

1 Introduction

Popular data mining techniques concern 0/1 data analysis by means of set patterns (i.e., frequent sets, association rules, closed sets, formal concepts). The huge research effort of the last 10 years has given rise to efficient complete solvers, i.e., algorithms which can compute complete collections of the set patterns which satisfy user-defined constraints (e.g., minimal frequency, minimal confidence, closeness or maximality). It is however common that the considered raw data is available as matrices where we get numerical values for a collection of attributes describing a collection of objects. Therefore, using the efficient techniques in 0/1 data has to start by Boolean property encoding, i.e., the computation of Boolean values for new sets of attributes. For instance, raw microarray data can be considered as a matrix whose rows denote biological samples and columns denote genes. In that context, each cell of the matrix is a quantitative measure of the activity of a given gene in a given biological sample. Several researchers have considered how to encode Boolean gene expression properties like, e.g., the gene over-expression [1, 11]. In such papers, the computed Boolean

matrix has the same number of attributes than the raw data but it encodes only one specific property. Efficient techniques like association rule mining (see, e.g., [1, 7]) or formal concept discovery (see, e.g., [4]) have been considered.

Such a Boolean encoding phase is however tedious. For instance, we still lack from a consensus on how the over-expression property of a gene can be specified or assessed. As a result, different views on over-expression will lead to different Boolean encoding and thus potentially quite different collections of patterns. To overcome these problems, we investigate the possibility to mine directly the numerical data in order to find interesting local patterns. Global pattern mining from numerical data, e.g., clustering and bi-clustering, has been extensively studied (see [10] for a survey). Heuristic search for local patterns has been studied as well (see, e.g., [2]). However, very few researchers have investigated the non heuristic, say complete, search of well-specified local patterns from numerical data. In this paper, we introduce the Numerical Bi-Sets as a new pattern domain (NBS). Intuitively, we specify collections of bi-sets, i.e., associated sets of rows and columns such that the specified cells (for each row-column pair) of the matrix contain similar values. This property is formalized in terms of constraints, and we provide a complete solver for computing NBS patterns. We start from a recent formalization of constraint-based bi-set mining from 0/1 data (extension of formal concepts towards fault-tolerance introduced in [3]) both for the design of the pattern domain and its associated solver. The next section concerns the formalization of the NBS pattern domain and its properties. Section 3 sketches our algorithm and Section 4 provides preliminary experimental results. Section 5 discusses related work and, finally, Section 6 concludes.

2 A new pattern domain for numerical data analysis

Let us consider a set of objects $\mathcal{O}$ and a set of properties $\mathcal{P}$ such that $|\mathcal{O}| = n$ and $|\mathcal{P}| = m$. Let us denote by $\mathcal{M}$ a real valued matrix of dimension $n \times m$ such that $\mathcal{M}(i, j)$ denotes the value of property $j \in \mathcal{P}$ for the object $i \in \mathcal{O}$ (see an example in Table 1). Our language of patterns is the language of bi-sets, i.e., couples made of a set of rows (objects) and a set of columns (properties). Intuitively, a bi-set (X, Y) with $X \in 2^{\mathcal{O}}$ and $Y \in 2^{\mathcal{P}}$ can be considered as a rectangle or sub-matrix within $\mathcal{M}$ modulo row and column permutations.

Definition 1 (NBS). *Numerical Bi-Sets (or* NBS *patterns) in a matrix are the bi-sets* (X, Y) *such that* $|X| \geq 1$ *and* $|Y| \geq 1$ *(*$X \subseteq \mathcal{O}$, $Y \subseteq \mathcal{P}$*) which satisfy the constraint* $\mathcal{C}_{in} \wedge \mathcal{C}_{out}$*:*

$$\mathcal{C}_{in}(X, Y) \equiv |\max_{i \in X,\, j \in Y} \mathcal{M}(i, j) - \min_{i \in X,\, j \in Y} \mathcal{M}(i, j)| \leq \epsilon$$

$$\mathcal{C}_{out}(X, Y) \equiv \forall y \in \mathcal{P} \setminus Y,\ |\max_{i \in X,\, j \in Y \cup \{y\}} \mathcal{M}(i, j) - \min_{i \in X,\, j \in Y \cup \{y\}} \mathcal{M}(i, j)| > \epsilon$$

$$\forall x \in \mathcal{O} \setminus X,\ |\max_{i \in X \cup \{x\},\, j \in Y} \mathcal{M}(i, j) - \min_{i \in X \cup \{x\},\, j \in Y} \mathcal{M}(i, j)| > \epsilon$$

where ϵ *is a user-defined parameter.*

Such bi-sets define a sub-matrix $\mathcal{S}$ of $\mathcal{M}$ such that the absolute value of the difference between the maximum value and the minimum value on $\mathcal{S}$ is less or equal to ϵ (see $\mathcal{C}_{in}$). Furthermore, none object or property can be added to the bi-set without violating this constraint (see $\mathcal{C}_{out}$). This ensures the maximality of the specified bi-sets.

	p_1	p_2	p_3	p_4	p_5
o_1	1	2	2	1	6
o_2	2	1	1	0	6
o_3	2	2	1	7	6
o_4	8	9	2	6	7

Table 1. A toy example of numerical data

In Figure 1 (left), we can find the complete collection of NBS patterns which hold in the data from Table 1 when we have $\epsilon = 1$. In Table 1, the two black rectangles are two examples of such NBS patterns (i.e., the underlined patterns of Figure 1 (left)). Figure 1 (right) is an alternative representation for them: each cross in the 3D-diagram denotes a row-column pair for the data from Table 1.

$((o_1, o_2, o_3, o_4), (p_5))$
$\underline{((o_3, o_4), (p_4, p_5))}$
$((o_4), (p_1, p_5))$
$((o_1, o_2, o_3, o_4), (p_3))$
$((o_4), (p_1, p_2))$
$((o_2), (p_2, p_3, p_4))$
$((o_1, o_2), (p_4))$
$((o_1), (p_1, p_2, p_3, p_4))$
$\underline{((o_1, o_2, o_3), (p_1, p_2, p_3))}$

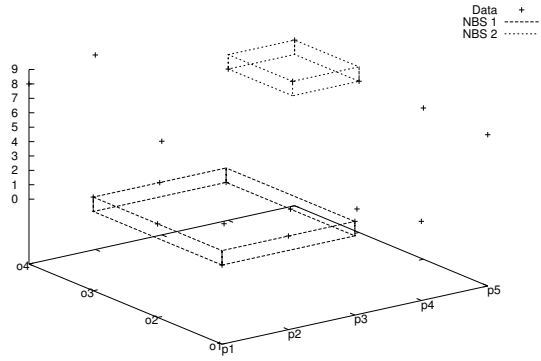


Fig. 1. Examples of NBS

The search space for bi-sets can be ordered thanks to a specialization relation.

Definition 2 (Specialization and monotonicity). *Our specialization relation on bi-sets denoted $\preceq$ is defined as follows: $(X_1, Y_1) \preceq (X_2, Y_2)$ iff $X_1 \subseteq X_2$ and $Y_1 \subseteq Y_2$. We say that (X_2, Y_2) extends or is an extension of (X_1, Y_1). A constraint $\mathcal{C}$ is anti-monotonic w.r.t. $\preceq$ iff $\forall B$ and $D \in 2^{\mathcal{O}} \times 2^{\mathcal{P}}$ s.t. $B \preceq D$, $\mathcal{C}(D) \Rightarrow \mathcal{C}(B)$. Dually, $\mathcal{C}$ is monotonic w.r.t. $\preceq$ iff $\mathcal{C}(B) \Rightarrow \mathcal{C}(D)$.*

Assume $\mathcal{W}_\epsilon$ denotes the whole collection of NBS patterns for a given threshold ϵ. Let us now discuss some interesting properties of this new pattern domain:

- $\mathcal{C}_{in}$ and $\mathcal{C}_{out}$ are respectively anti-monotonic and monotonic w.r.t. $\preceq$ (see property 1).
- Each NBS pattern (X, Y) from $\mathcal{W}_\epsilon$ is maximal w.r.t. $\preceq$ (see Property 2).
- If there exists a bi-set (X, Y) with similar values (belonging to an interval of size ϵ), then there exists a NBS (X', Y') from $\mathcal{W}_\epsilon$ such that $(X, Y) \preceq (X', Y\prime)$ (see Property 3).
- When ϵ increases, the size of NBS pattern increases too, whereas some new NBS patterns which are not extensions of previous one can appear (see Property 4).
- The collection of numerical bi-sets is paving the dataset (see Corollary 1), i.e., any data item belongs to at least one NBS pattern.

Property 1 (Monotonicity). The constraint $\mathcal{C}_{in}$ is anti-monotonic and the constraint $\mathcal{C}_{out}$ is monotonic.

Proof. Let (X, Y) a bi-set s.t. $\mathcal{C}_{in}(X, Y)$ is true, and let (X', Y') be a bi-set s.t. $(X', Y') \preceq (X, Y)$. It means that $\mathcal{C}_{in}(X', Y')$ is also true:

$$\begin{aligned} &|\max_{i\in X',\, j\in Y'} \mathcal{M}(i,j) - \min_{i\in X',\, j\in Y'} \mathcal{M}(i,j)| \\ &\leq |\max_{i\in X,\, j\in Y} \mathcal{M}(i,j) - \min_{i\in X,\, j\in Y} \mathcal{M}(i,j)| \leq \epsilon \end{aligned}$$

If (X, Y) satisfies $\mathcal{C}_{out}$ and $(X, Y) \preceq (X', Y')$, then $\mathcal{C}_{out}(X', Y')$ is also true:

$$\begin{aligned} &\forall y \in \mathcal{P} \setminus Y,\ |\max_{i\in X,\, j\in Y\cup\{y\}} \mathcal{M}(i,j) - \min_{i\in X,\, j\in Y\cup\{y\}} \mathcal{M}(i,j)| \\ &> \forall y \in \mathcal{P} \setminus Y',\ |\max_{i\in X',\, j\in Y'\cup\{y\}} \mathcal{M}(i,j) - \min_{i\in X',\, j\in Y'\cup\{y\}} \mathcal{M}(i,j)| > \epsilon \end{aligned}$$

Property 2 (Maximality). The NBS patterns are maximal bi-sets w.r.t. our specialization relation $\preceq$, i.e., if (X, Y_1) and (X, Y_2) are two NBS patterns from $\mathcal{W}_\epsilon$, then $Y_1 \not\subseteq Y_2$ and $Y_2 \not\subseteq Y_1$.

Proof. Assume $Y_1 \subseteq Y_2$. (X, Y_1) does not satisfy Equation 2, because for $y \in Y_2 \setminus Y_1$, $|\max_{i\in X} \mathcal{M}(i, y) - \min_{i\in X} \mathcal{M}(i, y)| \leq \epsilon$.

Property 3 (NBS patterns extending bi-sets of close values). Let $I_1, I_2 \in \mathbb{R}$, $I_1 \leq I_2$, and (X, Y) be a bi-set such that $\forall i \in X$, $\forall j \in Y$, $\mathcal{M}(i, j) \in [I_1, I_2]$. Then, there exists (U, V) a NBS with $\epsilon = |I_1 - I_2|$ such that $X \subseteq U$ and $Y \subseteq V$.

Thus, if there are bi-sets containing close values, there exists at least one NBS pattern which extends it.

Proof. V can be recursively constructed from $Y' = Y$ by adding a property y s.t. $y \in \mathcal{P} \setminus Y'$ to Y' if $|\max_{i\in X,\, j\in Y'\cup\{y\}} \mathcal{M}(i, j) - \min_{i\in X,\, j\in Y'\cup\{y\}} \mathcal{M}(i, j)| \leq \epsilon$, and then continue until none property can be added. At the end, $Y' = V$. After that, we extend in a similar way the set X towards U. By construction, (U, V) is a NBS pattern with $\epsilon = |I_1 - I_2|$. Notice that we can have several (U, V) which extend (X, Y).

When $\epsilon = 0$, the NBS pattern collection contains all maximal bi-sets of identical values. As a result, we get a paving (with overlapping) of the whole dataset.

Property 4 (NBS pattern size is growing with ϵ). Let (X, Y) be a NBS pattern from $\mathcal{W}_\epsilon$. Then there exists $(X', Y') \in \mathcal{W}_{\epsilon'}$ with $\epsilon' > \epsilon$ such that $X \subseteq X'$ and $Y \subseteq Y'$.

Proof. Proof is trivial given Property 3.

Corollary 1. *As $\mathcal{W}_0$ is paving the data, then $\mathcal{W}_\epsilon$ is paving the data as well.*

3 Algorithm

The whole collection of bi-sets ordered by $\preceq$ forms a lattice whose bottom is $(\bot_O, \bot_P) = (\emptyset, \emptyset)$ and top is $(\top_O, \top_P) = (\mathcal{O}, \mathcal{P})$. Let us note by $\mathcal{B}$ the set of sublattices[4] of $((\emptyset, \emptyset), (\mathcal{O}, \mathcal{P}))$: $\mathcal{B} = \{((X_1, Y_1), (X_2, Y_2))$ s.t. $X_1, X_2 \in 2^{\mathcal{O}}, Y_1, Y_2 \in 2^{\mathcal{P}}$ and $X_1 \subseteq X_2$, $Y_1 \subseteq Y_2\}$ where the first (resp. the second) bi-set is the bottom (resp. the top) element. The algorithm NBS-Miner explores some of the sublattices of $\mathcal{B}$ built by means of three mechanisms: enumeration, pruning and propagation.

- **Enumeration:** Let $Enum : \mathcal{B} \times \mathcal{O} \cup \mathcal{P} \rightarrow \mathcal{B}^2$ such that

$$Enum(((\bot_O, \bot_P), (\top_O, \top_P)), e) = \begin{cases} (((\bot_O \cup \{e\}, \bot_P), (\top_O, \top_P)), ((\bot_O, \bot_P), (\top_O \setminus \{e\}, \top_P))) \text{ if } e \in \mathcal{O} \\ (((\bot_O, \bot_P \cup \{e\}), (\top_O, \top_P)),\ ((\bot_O, \bot_P), (\top_O, \top_P \setminus \{e\}))) \text{ if } e \in \mathcal{P} \end{cases}$$

  where $e \in \top_O \setminus \bot_O$ or $e \in \top_P \setminus \bot_P$. *Enum* generates two new sublattices which are a partition of its input parameter.
  Let $Choose : \mathcal{B} \rightarrow \mathcal{O} \cup \mathcal{P}$ be a function which returns one of the element $e \in \top_O \setminus \bot_O \cup \top_P \setminus \bot_P$.
- **Pruning:** Let $Prune^m_{\mathcal{C}} : \mathcal{B} \rightarrow \{\text{TRUE,FALSE}\}$ be a function which returns True iff the monotonic constraint $\mathcal{C}^m$ (w.r.t. $\preceq$) is satisfied by the top of the sublattice.

$$Prune^m_{\mathcal{C}}((\bot_O, \bot_P), (\top_O, \top_P)) \equiv \mathcal{C}^m(\top_O, \top_P)$$

  If $Prune^m_{\mathcal{C}}((\bot_O, \bot_P), (\top_O, \top_P))$ is false then none bi-set contained in the sublattice satisfies $\mathcal{C}^m$.
  Let $Prune^{am}_{\mathcal{C}} : \mathcal{B} \rightarrow \{\text{TRUE,FALSE}\}$ be a function which returns True iff the anti-monotonic constraint $\mathcal{C}^{am}$ (w.r.t $\preceq$) is satisfied by te bottom of the sublattice:

$$Prune^{am}_{\mathcal{C}}((\bot_G, \bot_M), (\top_G, \top_M)) \equiv \mathcal{C}^{am}(\bot_G, \bot_M)$$

[4] X is a sublattice of Y if Y is a lattice, X is a subset of Y and X is a lattice with the same join and meet operations as Y.

If $Prune^{am}_{\mathcal{C}}((\bot_O, \bot_P), (\top_O, \top_P))$ is false then none bi-set contained in the sublattice satisfies $\mathcal{C}^{am}$.
Let $Prune_{\mathcal{C}_{NBS}} : \mathcal{B} \rightarrow \{\text{TRUE,FALSE}\}$ be the pruning function. Due to property 1, we have

$$Prune_{\mathcal{C}_{NBS}}((\bot_O, \bot_P), (\top_O, \top_P)) \equiv \mathcal{C}_{in}(\bot_O, \bot_P) \wedge \mathcal{C}_{out}(\top_O, \top_P)$$

When $Prune_{\mathcal{C}_{NBS}}((\bot_O, \bot_P), (\top_O, \top_P))$ is false then no NBS are contained in the sublattice $((\bot_O, \bot_P), (\top_O, \top_P))$.

– **Propagation:** $\mathcal{C}_{out}$ can be used to reduce the size of the sublattices by moving objects of $\top_O \setminus \bot_O$ into $\bot_O$ or outside $\top_O$, and similarly on attributes. The function $Prop_{in}\, \mathcal{B} \rightarrow \mathcal{B}$ and $Prop_{out}\, \mathcal{B} \rightarrow \mathcal{B}$ are used to do it as follow:

$$Prop_{in}((\bot_O, \bot_P), (\top_O, \top_P)) = \{((\bot^1_O, \bot^1_P), (\top_O, \top_P)) \in \mathcal{B} \mid$$
$$\bot^1_O = \bot_O \cup \{x \in \top_O \setminus \bot_O \mid \mathcal{C}_{out}((\bot_O, \bot_P), (\top_O \cup \{x\}, \top_P)) \text{ is false}\}$$
$$\bot^1_P = \bot_P \cup \{x \in \top_P \setminus \bot_P \mid \mathcal{C}_{out}((\bot_O, \bot_P), (\top_O, \top_P \cup \{x\})) \text{ is false}\}$$

$$Prop_{out}((\bot_O, \bot_P), (\top_O, \top_P)) = \{((\bot_O, \bot_P), (\top^1_O, \top^1_P)) \in \mathcal{B} \mid$$
$$\top^1_O = \top_O \setminus \{x \in \top_G \setminus \bot_G \mid \mathcal{C}_{in}((\bot_O, \bot_P), (\top_O \cup \{x\}, \top_P)) \text{ is false}\}$$
$$\top^1_P = \top_P \setminus \{x \in \top_P \setminus \bot_P \mid \mathcal{C}_{in}((\bot_O, \bot_P), (\top_O, \top_P \cup \{x\})) \text{ is false}\}$$

Let $Prop\, \mathcal{B} \rightarrow \mathcal{B}$ s.t. $Prop_{in}(Prop_{out}(\mathcal{L}))$ is recursively applied as long as its result changes.

We call a leaf a sublattice $\mathcal{L} = ((\bot_O, \bot_P), (\top_O, \top_P))$ which contains only one bi-set i.e., $(\bot_O, \bot_P) = (\top_O, \top_P)$. DR-bi-sets are these leaves.

4 Experimentations

We report a preliminary experimental evaluation of the NBS pattern domain and its implemented solver. We have been considering the "peaks" matrix of matlab (30*30 matrix with values ranged between -10 and +9). We used $\epsilon = 4.5$ and we obtained 1700 NBS patterns. On Figure 2 (left), we plot in white one extracted NBS. The two axes ranged from 0 to 30 correspond to the two matrix dimensions and the third one indicates their corresponding values (row-column pairs).

In a second experiment, we enforced that the values inside the extracted patterns to be greater than 1.95 (minimal value constraint). The Figure 2 (right) shows the 228 extracted NBS patterns when $\epsilon = 0.1$. Indeed, the white area corresponds to the union of 228 extracted patterns.

To study the impact of ϵ parameter, we used the malaria dataset [5]. It concerns the numerical gene expression value of 3 719 genes of P. falciparum during its complete lifecycle (a time series of 46 biological situations). We used a minimal size constraint on both dimension, i.e., looking for the NBS patterns (X, Y)

$\mathcal{M}$ is a real valued matrix, $\mathcal{C}$ a conjunction of monotonic and anti-monotonic constraints on $2^{\mathcal{O}} \times 2^{\mathcal{P}}$ and ϵ is a positive value.

NBS-Miner

Generate$((\emptyset, \emptyset), (\mathcal{O}, \mathcal{P}))$

End NBS-Miner
Generate$(\mathcal{L})$

Let $\mathcal{L} = ((\bot_O, \bot_P), (\top_O, \top_P))$
$\mathcal{L} \leftarrow Prop(\mathcal{L})$
If $Prune(\mathcal{L})$ then
  If $(\bot_O, \bot_P) \neq (\top_O, \top_P)$ then
    $(\mathcal{L}_1, \mathcal{L}_2) \leftarrow Enum(\mathcal{L}, Choose(\mathcal{L}))$
    Generate$(\mathcal{L}_1)$
    Generate$(\mathcal{L}_2)$
  Else Store $\mathcal{L}$
  End if
End if

End Generate

Table 2. NBS-Miner pseudo-code

s.t. $|X| > 4$ and $|Y| > 4$. Furthermore, we have been adding a minimal value constraint. Figure 3 provides the mean and standard deviation of the area of the NBS patterns from this dataset w.r.t. the ϵ value.

As it was expected owed to Property 4, the mean area increases with ϵ.

Figure 4 reports on the number of NBS patterns in the malaria dataset. From $\epsilon = 75$ to $\epsilon = 300$, this number decreases. It shows that the size of the NBS pattern collection tends to decrease when ϵ increases. Intuitively, many patterns are gathered when ϵ increases whereas few patterns are extended by generating more than one new pattern. Moreover, the minimal size constraint can explain the increase of the collection size. Finally, when the pattern size increases with ϵ, new NBS patterns can appear in the collection.

5 Related work

[13, 6, 12] propose to extend classical frequent itemset and association rule definitions for numerical data. In [13], the authors generalize the classical notion of itemset support in 0/1 data when considering other data types, e.g., numerical ones. Support computation requires data normalization, first translating the values to be positive, and then dividing each column entry by the sum of the column entries. After such a treatment, each entry is between 0 and 1, and the sum of the values for a column is equal to 1. The support of an itemset is then computed

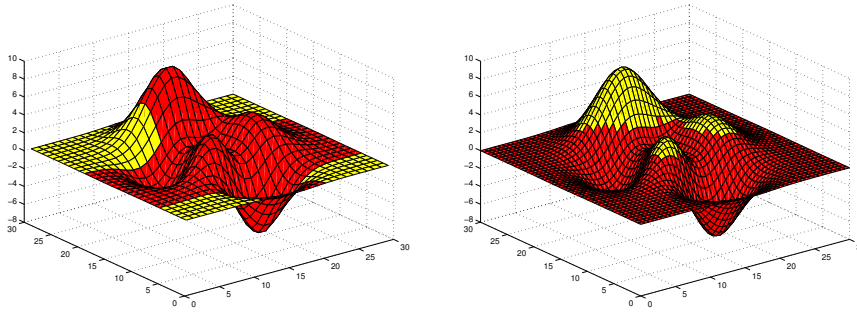


Fig. 2. Examples of extracted NBS

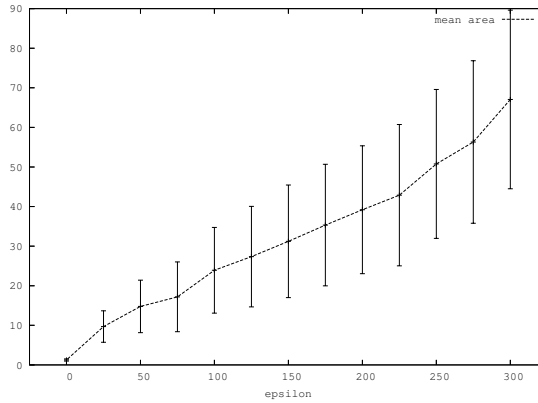


Fig. 3. Mean area of the NBS w.r.t. ϵ

as the sum on each row of the minimum of the entries of this itemset. If the items have identical values on all the rows, then the support is equal to 1, and the more the items are different, the more the support value decreases toward 0. This support function is anti-monotonic, and thus the authors propose to adapt an Apriori algorithm to compute the frequent itemsets according to this new support definition. [6] proposes new methods to measure the support of itemsets in numerical data and categorical data. They adapt three well-known correlation measures: Kendall's τ, Spearman's ρ and Spearman's Footrule F. These measures are based on the rank of the values of objects for each attribute, not the values themselves. They extend these measures to sets of attributes (instead of 2 variables). Efficient algorithms are proposed. [12] uses an optimization setting for finding association rules in numerical data. The type of extracted association rules is: "if the weighted sum of some variables is greater than a threshold then a different weighted sum of variables is with high probability greater than a second threshold". They propose to use hyperplanes to represent the left-hand and the

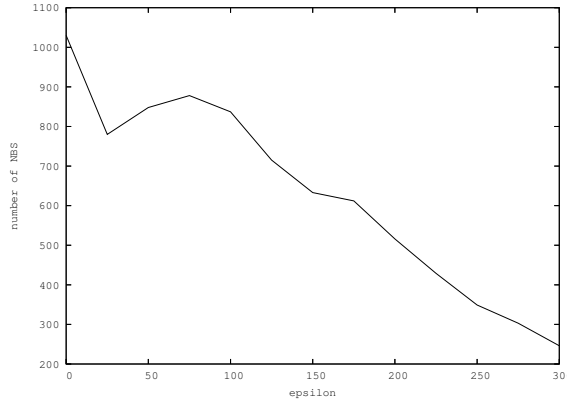


Fig. 4. Collection sizes w.r.t. ϵ.

right-hand sides of such rules. Confidence and coverage measures are used. It is unclear wether it is possible to extend these approaches to bi-set computation.

Hartigan proposes a bi-clustering algorithm that can be considered as a specific collection of bi-sets [8]. He introduced a partition-based algorithm called "Block Clustering". It splits the original data matrix into bi-sets and it uses the variance of the values inside the bi-sets to evaluate the quality of each bi-set. Then, a so-called ideal constant cluster has a variance equal to zero. To avoid the partitioning of the dataset into bi-sets with only one row and one column (i.e., leading to ideal clusters), the algorithm searches for K bi-sets within the data. The quality of a collection of K bi-sets is considered as the sum of the variance of the K bi-sets. Unfortunately, this approach uses local optimization procedure which can lead to unstable results.

In [14], the authors propose a method to isolate subspace clusters (bi-sets) containing objects varying similarly on subset of columns. They propose to compute bi-sets (X, Y) such that given $a, b \in X$ and $c, d \in Y$ the 2×2 sub-matrix entries $((a, b), (c, d))$ included in (X, Y) satisfies $|\mathcal{M}(a, c) + \mathcal{M}(b, d) - (\mathcal{M}(a, d) + \mathcal{M}(b, c))| \leq \delta$. Intuitively, this constraint enforces that the change of value on the two attributes between the two objects is confined by δ. Thus, inside the bi-sets, the values have the same profile. The algorithm first considers all pairs of objects and all pairs of attributes, and then combines them to compute all the bi-sets satisfying the anti-monotonic constraint.

Liu and Wang [9] have proposed an exhaustive bi-cluster enumeration algorithm. Since they are looking for order-preserving bi-sets with a minimum number of rows and a minimum number of columns. It means that for each extracted bi-set (X, Y), it exists an order on Y such that according to this order and for each element of X the values are increasing. They want to provide all the bi-clusters that, after column reordering, represent coherent evolutions of the symbols in the matrix. It is achieved by using a pattern discovery algorithm

heavily inspired in sequential pattern mining algorithms. These two local pattern types are well defined and efficient solvers are proposed. Notice however that these patterns are not symmetrical: they capture similar variations on one dimension and not similar values.

Except the bi-clustering method of [8], all these methods focus on one of the two dimensions. We have proposed to compute bi-sets with a symmetrical definition which is one of the main difficulties in bi-set mining. This is indeed one of the lessons from all the previous work on bi-set mining from 0/1 data, and, among others, the several attempts to mine fault-tolerant extensions to formal concepts instead of fault-tolerant itemsets [3].

6 Conclusion

Efficient data mining techniques concern 0/1 data analysis by means of set patterns. It is however common, for instance in the context of gene expression data analysis, that the considered raw data is available as a collection of real numbers. Therefore, using the available algorithms needs for a beforehand Boolean property encoding. To overcome such a tedious task, we started to investigate the possibility to mine set patterns directly from the numerical data. We introduced the Numerical Bi-Sets as a new pattern domain. Some nice properties of NBS patterns have been considered. We have described our implemented solver NBS-Miner in quite generic terms, i.e., emphasizing the fundamental operations for the complete computation of NBS patterns. Notice also that other monotonic or anti-monotonic constraints can be used in conjunction with $\mathcal{C}_{in} \wedge \mathcal{C}_{out}$, i.e., the constraint which specifies the pattern domain. It means that search space pruning can be enhanced for mining real-life datasets provided that further user-defined constraints are given. The perspectives are obviously related to further experimental validation, especially the study of scalability issues. Furthermore, we still need for an in-depth understanding of the complementarity between NBS pattern mining and bi-set mining from 0/1 data.

Acknowledgments. This research is partially funded by the EU contract IQ FP6-516169 (FET arm of the IST programme). J. Besson is paid by INRA (ASC post-doc).

References

1. C. Becquet, S. Blachon, B. Jeudy, J.-F. Boulicaut, and O. Gandrillon. Strong-association-rule mining for large-scale gene-expression data analysis: a case study on human sage data. *Genome Biology*, 12, November 2002.
2. S. Bergmann, J. Ihmels, and N. Barkai. Iterative signature algorithm for the analysis of large-scale gene expression data. *Physical Review*, 67, March 2003.
3. J. Besson, R. Pensa, C. Robardet, and J.-F. Boulicaut. Constraint-based mining of fault-tolerant patterns from boolean data. In *Revised Selected and Invited Papers KDID'05*, volume 3933 of *LNCS*, pages 55–71. Springer-Verlag, 2006.

4. J. Besson, C. Robardet, J.-F. Boulicaut, and S. Rome. Constraint-based concept mining and its application to microarray data analysis. *Intelligent Data Analysis*, 9(1):59–82, 2005.
5. Z. Bozdech, M. Llinás, B. Pulliam, E. Wong, J. Zhu, and J. DeRisi. The transcriptome of the intraerythrocytic developmental cycle of plasmodium falciparum. *PLoS Biology*, 1(1):1–16, 2003.
6. T. Calders, B. Goethals, and S. Jaroszewicz. Mining rank correlated sets of numerical attributes. In *Proceedings ACM SIGKDD'06*. To appear.
7. C. Creighton and S. Hanash. Mining gene expression databases for association rules. *Bioinformatics*, 19(1):79–86, November 2002.
8. J. Hartigan. Direct clustering of data matrix. *Journal of the American Statistical Association*, 67(337):123–129, March 1972.
9. J. Liu and W. Wang. Op-cluster: Clustering by tendency in high dimensional space. In *Proceedings IEEE ICDM'03*, pages 187–194, Melbourne, USA, Dec. 2003.
10. S. C. Madeira and A. L. Oliveira. Biclustering algorithms for biological data analysis: A survey. *ACM/IEEE Trans. on computational biology and bioinformatics*, 1(1):24–45, 2004.
11. R. G. Pensa, C. Leschi, J. Besson, and J.-F. Boulicaut. Assessment of discretization techniques for relevant pattern discovery from gene expression data. In *Proceedings ACM BIOKDD'04*, pages 24–30, Seattle, USA, August 2004.
12. U. Ruckert, L. Richter, and S. Kramer. Quantitative association rules based on half-spaces: An optimization approach. In *Proceedings IEEE ICDM'04*, pages 507–510, Brighton, UK, Nov. 2004.
13. M. Steinbach, P.-N. Tan, H. Xiong, and V. Kumar. Generalizing the notion of support. In *Proceedings ACM SIGKDD'04*, pages 689–694, Seatle, USA, 2004.
14. H. Wang, W. Wang, J. Yang, and P. S. Yu. Clustering by pattern similarity in large data sets. In *Proceedings ACM SIGMOD'02*, pages 394–405, Madison, USA, June 2002.

Weighted and Probabilistic Instances of the Soft Constraint Based Pattern Mining Paradigm

Stefano Bistarelli[1,2] and Francesco Bonchi[3]

[1] Dipartimento di Scienze, Università degli Studi "G. D'Annunzio", Pescara, Italy
[2] Istituto di Informatica e Telematica, CNR, Pisa, Italy
[3] Pisa KDD Laboratory, ISTI - C.N.R., Pisa, Italy
e-mail: bista@sci.unich.it; francesco.bonchi@isti.cnr.it

Abstract. The paradigm of pattern discovery based on constraints has been recognized as a core technique in inductive querying: constraints provide to the user a tool to drive the discovery process towards potentially *interesting* patterns, with the positive side effect of achieving a more efficient computation. However, due to the lack of research on methodological issues, the constraint-based pattern mining framework still suffers from many problems which limit its practical relevance. In our previous work [4], we analyzed such limitations and showed how they flow out from the same source: the fact that in the classical constraint-based mining, a constraint is a rigid boolean function which returns either *true* or *false*. To overcome such limitations we introduced the new paradigm of pattern discovery based on *Soft Constraints*, and instantiated our idea to the fuzzy soft constraints. In this paper we extend the framework to deal with probabilistic and weighted soft constraints: we provide theoretical basis and detailed experimental analysis. Finally we discuss of how deal with *top-k* queries.

1 Introduction

The paradigm of pattern discovery based on constraints was introduced with the aim of providing to the user a tool to drive the discovery process towards potentially *interesting* patterns, with the positive side effect of achieving a more efficient computation. So far the research on this paradigm has mainly focused on the latter aspect: the study of constraints properties and, on the basis of these properties, the development of efficient algorithms for the evaluation of constraint-based mining queries. Despite such algorithmic research effort, and regardless some successful applications, e.g., in medical domain [8, 11], or in biological domain [3], the constraint-based pattern mining framework still suffers from many problems which limit its practical relevance. In our previous work [4], we analyzed such limitations and showed how they flow out from the same source: the fact that in the classical constraint-based mining, a constraint is a rigid boolean function which returns either *true* or *false*. Indeed, interestingness is not a dichotomy. Following this consideration, we introduced in [4] the new paradigm of pattern discovery based on *Soft Constraints*, where constraints are no longer rigid boolean functions. In particular we adopted a definition of soft constraints based on the mathematical concept of *semiring*. Albeit based on a simple idea, our proposal has the merit of providing a rigorous theoretical framework, which is very general (having the classical paradigm as a particular instance), and which overcomes all the major

methodological drawbacks of the classical constraint-based paradigm, representing a step further towards practical pattern discovery.

While in our previous paper we instantiated the framework to the *fuzzy* semiring, in this paper we extend the framework to deal with the *probabilistic* and the *weighted* semirings: these different constraints instances can be used to model different situations, depending on the application at hand. We provide the formal problem definition and the theoretical basis to develop concrete solvers for the mining problems we defined. In particular, we will show how to build a concrete *soft-constraint based pattern discovery system*, by means of a set of appropriate wrappers around a crisp constraint pattern mining system. The mining system for classical constraint-based pattern discover that we adopted is CONQUEST, a system which we have developed at Pisa KDD Laboratory [7]. Such a system is based on a mining engine which is a general Apriori-like algorithm which, by means of *data reduction* and *search space pruning*, is able to push a wide variety of constraints (practically all possible kinds of constraints which have been studied and characterized) into the frequent itemsets computation. Finally, we discuss how to answer to *top-k* queries.

2 Soft Constraint Based Pattern Mining

Several formalizations of the concept of soft constraints are currently available. In the following, we refer to the formalization based on *c-semirings* [6]: a semiring-based constraint assigns to each instantiation of its variables an associated value from a partially ordered set. When dealing with crisp constraints, the values are the boolean $true$ and $false$ representing the admissible and/or non-admissible values; when dealing with soft constraints the values are interpreted as preferences/costs. The framework must also handle the combination of constraints. To do this one must take into account such additional values, and thus the formalism must provide suitable operations for combination ($\times$) and comparison ($+$) of tuples of values and constraints. This is why this formalization is based on the mathematical concept of semiring.

Definition 1 (c-semirings [6]). *A semiring is a tuple $\langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$ such that: A is a set and $\mathbf{0}, \mathbf{1} \in A$; $+$ is commutative, associative and $\mathbf{0}$ is its unit element; $\times$ is associative, distributes over $+$, $\mathbf{1}$ is its unit element and $\mathbf{0}$ is its absorbing element. A c-semiring ("c" stands for "constraint-based") is a semiring $\langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$ such that $+$ is idempotent with $\mathbf{1}$ as its absorbing element and $\times$ is commutative.*

Definition 2 (soft constraint on c-semiring [6]). *Given a c-semiring $S = \langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$ and an ordered set of variables V over a finite domain D, a constraint is a function which, given an assignment $\eta : V \rightarrow D$ of the variables, returns a value of the c-semiring. By using this notation we define $\mathcal{C} = \eta \rightarrow A$ as the set of all possible constraints that can be built starting from S, D and V.*

In the following we will always use the word semiring as standing for c-semiring.

Example 1. The following example illustrates the definition of soft constraint based on semiring, using the example mining query:

$$\mathcal{Q}: \ supp_{\mathcal{D}}(X) \geq 1500 \ \wedge \ avg(X.weight) \leq 5 \ \wedge \ sum(X.price) \geq 20$$

which requires to mine, from database $\mathcal{D}$, all patterns which are frequent (have a support at least 1500), have average weight at most 5 and a sum of prices at least 20. In this

context, we have that the ordered set of variables V is $\langle supp_{\mathcal{D}}(X), avg(X.weight), sum(X.price)\rangle$; the domain D is: $D(supp_{\mathcal{D}}(X)) = \mathbb{N}$, $D(avg(X.weight)) = \mathbb{R}^+$, and $D(sum(X.price)) = \mathbb{N}$. If we consider the classical crisp framework (i.e., hard constraints) we are on the boolean semiring: $S_{Bool} = \langle\{true, false\}, \vee, \wedge, false, true\rangle$. A soft constraint C is a function $V \to D \to A$; e.g., $supp_{\mathcal{D}}(X) \to 1700 \to true$.

The $+$ operator is what we use to compare tuples of values (or patterns, in our context). Let us consider the relation $\leq_S$ (where S stands for the specified semiring) over A such that $a \leq_S b$ iff $a + b = b$. It is possible to prove that: $\leq_S$ is a partial order; $+$ and $\times$ are monotone on $\leq_S$; $\mathbf{0}$ is its minimum and $\mathbf{1}$ its maximum, and $\langle A, \leq_S\rangle$ is a complete lattice with least upper bound operator $+$. In the context of pattern discovery $a \leq_S b$ means that the pattern b is *more interesting* than a, where interestingness is defined by a combination of soft constraints. When using (soft) constraints it is necessary to specify, via suitable combination operators, how the level of interest of a combination of constraints is obtained from the interest level of each constraint. The combined weight (or interest) of a combination of constraints is computed by using the operator $\otimes : \mathcal{C} \times \mathcal{C} \to \mathcal{C}$ defined as $(C_1 \otimes C_2)\eta = C_1\eta \times_S C_2\eta$.

Example 2. In this example, and in the rest of the paper, we use for the patterns the notation $p : \langle v_1, v_2, v_3\rangle$, where p is an itemset, and $\langle v_1, v_2, v_3\rangle$ denote the three values $\langle supp_{\mathcal{D}}(p), avg(p.weight), sum(p.price)\rangle$ corresponding to the three constraints in the conjunction in the query $\mathcal{Q}$ of Example 1. Consider, for instance, the following three patterns: $p_1 : \langle 1700, 0.8, 19\rangle$, $p_2 : \langle 1550, 4.8, 54\rangle$, $p_3 : \langle 1550, 2.2, 26\rangle$. If we adopt the classical crisp framework, in the mining query $\mathcal{Q}$ we have to combine the three constraints using the $\wedge$ operator (which is the $\times$ in the boolean semiring S_{Bool}). Consider for instance the pattern $p_1 : \langle 1700, 0.8, 19\rangle$ for the ordered set of variables $V = \langle supp_{\mathcal{D}}(X), avg(X.weight), sum(X.price)\rangle$. The first and the second constraint are satisfied leading to the semiring level $true$, while the third one is not satisfied and has associated level $false$. Combining the three values with $\wedge$ we obtain $true \wedge true \wedge false = false$ and we can conclude that the pattern $\langle 1700, 0.8, 19\rangle$ is not interesting w.r.t. our purposes. Similarly, we can instead compute level $true$ for pattern $p_3 : \langle 1550, 2.2, 26\rangle$ corresponding to an interest w.r.t. our goals.

However, dividing patterns in *interesting* and *non-interesting* is sometimes not meaningful nor useful. Most of the times we want to say that each pattern is interesting with a specific level of preference. This idea is at the basis of the soft constraint based pattern mining paradigm [4].

Definition 3 (Soft Constraint Based Pattern Mining). *Let $\mathcal{P}$ denote the domain of possible patterns. A soft constraint on patterns is a function $\mathcal{C} : \mathcal{P} \to A$ where A is the carrier set of a semiring $S = \langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$. Given a combination of soft constraints $\otimes\mathcal{C}$, i.e., a description of what is considered by the user an interesting pattern, we define two different problems:*

λ-interesting: *given a minimum interest threshold $\lambda \in A$, it is required to mine the set of all λ-interesting patterns, i.e., $\{p \in \mathcal{P} | \otimes \mathcal{C}(p) \geq_S \lambda\}$.*

top-k: *given a threshold $k \in \mathbb{N}$, it is required to mine the top-k patterns $p \in \mathcal{P}$ w.r.t. the order $\leq_S$.*

In the rest of the paper we adopt the notation $int_S^{\mathcal{P}}(\lambda)$ to denote the problem of mining λ-interesting patterns (from pattern domain $\mathcal{P}$) on the semiring S, and similarly $top_S^{\mathcal{P}}(k)$, for the corresponding top-k mining problem. Note that the Soft Constraint Based Pattern Mining paradigm just defined, has many degrees of freedom. In particular, it can be instantiated:

1. on the domain of patterns $\mathcal{P}$ in analysis (e.g., itemsets, sequences, trees or graphs),
2. on the semiring $S = \langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$ (e.g., boolean, fuzzy, weighted or probabilistic), and
3. on one of the two possible mining problems, i.e., λ-interesting or top-k mining.

In other terms, by means of Definition 3, we have defined many different mining problems: it is worth noting that the classical constraint based frequent itemsets mining, is just a particular instance of our framework. In particular, it corresponds to the mining of λ-interesting itemsets on the boolean semiring, where $\lambda = true$, i.e., $int_b^{\mathcal{I}}(true)$. In our previous paper [4] we have shown how to deal with the mining problem $int_f^{\mathcal{I}}(\lambda)$ (i.e., λ-interesting Itemsets on the Fuzzy Semiring), in this paper we show how to extend our framework to deal with (i) $int_p^{\mathcal{I}}(\lambda)$ (i.e., λ-interesting Itemsets on the Probabilistic Semiring), (ii) $int_w^{\mathcal{I}}(\lambda)$ (i.e., λ-interesting Itemsets on the Weighted Semiring), and (iii) mining top-k itemsets on any semiring.

The methodology we adopt is based on the property that in a c-semiring $S = \langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$ the $\times$-operator is *extensive* [6], i.e, $a \times b \leq_S a$ for all $a, b \in A$. Thanks to this property, we can easily prune away some patterns from the set of possibly interesting ones. In particular this result directly applies when we want to solve a λ-interesting problem. In fact for any semiring (fuzzy, weighted, probabilistic) we have that [6]:

Proposition 1. *Given a combination of soft constraints $\otimes\mathcal{C}$ based on a semiring S, for any pattern $p \in \mathcal{P}$: $\otimes\mathcal{C}(p) \geq_S \lambda \Rightarrow \forall C \in \otimes\mathcal{C} : C(p) \geq_S \lambda$.*

Proof Straightforward from the extensivity of $\times$.

Therefore, computing all the λ-interesting patterns can be done by solving a crisp problem where all the constraint instances with semiring level lower than λ have been assigned level $false$, and all the instances with semiring level greater or equal to λ have been assigned level $true$. In fact, if a pattern does not satisfy such conjunction of crisp constraints, it will not be neither interesting w.r.t. the soft constraints. Using this theoretical result, and some simple arithmetic we can transform each soft constraint in a corresponding crisp constraint, push the crisp constraint in the mining computation to prune uninteresting patterns, and when needed, post-process the solution of the crisp problem, to remove uninteresting patterns from it.

3 Mining $int_p^{\mathcal{I}}(\lambda)$ (λ-interesting Itemsets on the Probabilistic Semiring)

Using the probabilistic constraints framework [9] we suppose each constraint to have an independent probability law, and combination is computed performing the product of the semiring value of each constraint instantiations. As a result, the semiring corresponding to the probabilistic framework is $S_P = \langle [0, 1], max, \times, 0, 1\rangle$.

Consider the constraints graphical representations in Figure 1, where the semiring values between 0 and 1 are this time interpreted as probabilities. In this situation for

the pattern $p_1 = \langle 1700, 0.8, 19\rangle$ we obtain that: $C_1(p_1) = 0.83$, $C_2(p_1) = 1$ and $C_3(p_1) = 0.45$. Since in the probabilistic semiring the combination operator $\times$ is the arithmetic multiplication, we got that the interest level of p_1 is 0.37. Similarly for p_2 and p_3:

- $p_1 : C_1 \otimes C_2 \otimes C_3(1700, 0.8, 19) = \times(0.83, 1, 0.45) = 0.37$
- $p_2 : C_1 \otimes C_2 \otimes C_3(1550, 4.8, 54) = \times(0.58, 0.6, 1) = 0.35$
- $p_3 : C_1 \otimes C_2 \otimes C_3(1550, 2.2, 26) = \times(0.58, 1, 0.8) = 0.46$

Therefore, with this particular instance we got that $p_2 <_{S_P} p_1 <_{S_P} p_3$, i.e., p_3 is the most interesting pattern among the three. Dealing with the probabilistic semiring, we can readapt most of the framework developed for the fuzzy semiring. In fact the two semirings are based on the same set $[0, 1]$ and on the same $+$ operator which is max. The only distinguishing element is the $\times$ operator which is min for the fuzzy semiring, while it is the arithmetic *times* for the probabilistic semiring. This means that we can straightforwardly readapt the problem definition, the way of defining the behaviour of soft constraints, and the *crisp translation*.

Definition 4. *Let $\mathcal{I} = \{x_1, ..., x_n\}$ be a set of items, where an item is an object with some predefined attributes (e.g., price, type, etc.). A soft constraint on itemsets, based on the probabilistic semiring, is a function $\mathcal{C} : 2^{\mathcal{I}} \rightarrow [0, 1]$. Given a combination of such soft constraints $\otimes\mathcal{C} \equiv \mathcal{C}_1 \otimes \ldots \otimes \mathcal{C}_n$, we define the interest level of an itemset $X \in 2^{\mathcal{I}}$ as $\otimes\mathcal{C}(X) = \mathcal{C}_1(X) \times \cdots \times \mathcal{C}_n(X)$. Given a minimum interest threshold $\lambda \in]0, 1]$, the λ-interesting itemsets mining problem, requires to compute $int_p^{\mathcal{I}}(\lambda) = \{X \in 2^{\mathcal{I}} | \otimes\mathcal{C}(X) \geq \lambda\}$.*

Definition 5. *A soft constraint $\mathcal{C}$ on itemsets, based on the probabilistic semiring, is defined by a quintuple $\langle Agg, Att, \theta, t, \alpha\rangle$, where:*

- *$Agg \in \{supp, min, max, count, sum, range, avg, var, median, std, md\}$;*
- *Att is the name of the attribute on which the aggregate agg is computed (or the transaction database, in the case of the frequency constraint);*
- *$\theta \in \{\leq, \geq\}$;*
- *$t \in \mathbb{R}$ corresponds to the center of the interval and it is associated to the semiring value 0.5;*
- *$\alpha \in \mathbb{R}^+$ is the softness parameter, which defines the inclination of the preference function (and thus the width of the interval).*

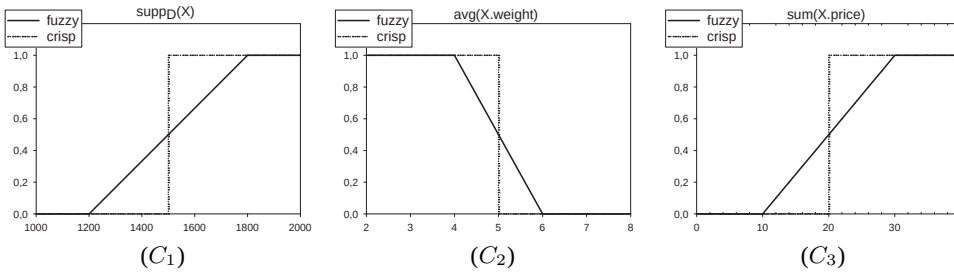


Fig. 1. Graphical representation of possible probabilistic instance of the constraints in the mining query $\mathcal{Q}$ in Example 1.

In particular, if $\theta = \leq$ (as in Figure $1(C_2)$) then $\mathcal{C}(X)$ is 1 for $X \leq (t - \alpha t)$, is 0 for $X \geq (t + \alpha t)$, and is linearly decreasing from 1 to 0 within the interval $[t - \alpha t, t + \alpha t]$. The other way around if $\theta = \geq$ (as, for instance, in Figure $1(C_3)$). Note that if the softness parameter α is 0, then we obtain the crisp (or hard) version of the constraint.

Example 3. Consider again the query $\mathcal{Q}$ given in Example 1, and its probabilistic instance graphically described by Figure 1. Such query can be expressed in our constraint language as:

$$\langle supp, \mathcal{D}, \geq, 1500, 0.2\rangle, \langle avg, weight, \leq, 5, 0.2\rangle, \langle sum, price, \geq, 20, 0.5\rangle$$

Definition 6. *Given a probabilistic soft constraint $\mathcal{C} \equiv \langle Agg, Att, \theta, t, \alpha\rangle$, and a minimum interest threshold λ, we define the crisp translation of $\mathcal{C}$ w.r.t. λ as:*

$$\mathcal{C}^{\lambda}_{crisp} \equiv \begin{cases} Agg(Att) \geq t - \alpha t + 2\lambda\alpha t, & \text{if } \theta = \geq \\ Agg(Att) \leq t + \alpha t - 2\lambda\alpha t, & \text{if } \theta = \leq \end{cases}$$

In [4] we proved that, on the fuzzy semiring, given a combination of soft constraints $\otimes\mathcal{C} \equiv \mathcal{C}_1 \otimes \ldots \otimes \mathcal{C}_n$, and a minimum interest threshold λ, if we consider the conjunction of crisp constraints obtained by conjoining the crisp translation of each constraint in $\otimes\mathcal{C}$ w.r.t. λ (i.e., $\mathcal{C}' \equiv {\mathcal{C}_1}^{\lambda}_{crisp} \wedge \ldots \wedge {\mathcal{C}_n}^{\lambda}_{crisp}$), it holds that

$$int^{\mathcal{I}}_f(\lambda) = \{X \in 2^{\mathcal{I}} | \otimes \mathcal{C}(X) \geq \lambda\} = Th(\mathcal{C}')$$

Similarly, the following property holds:

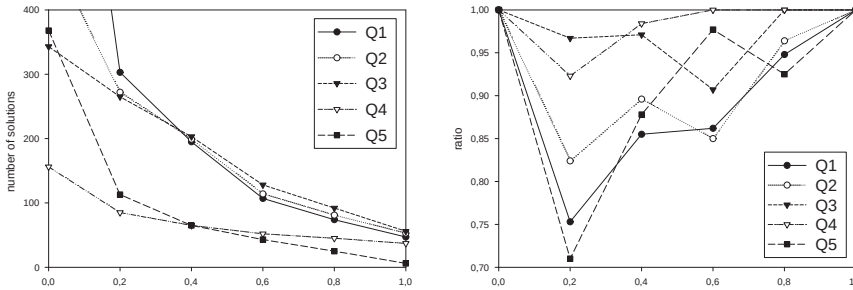
Proposition 2. *Given the vocabulary of items $\mathcal{I}$, a combination of soft constraints $\otimes\mathcal{C} \equiv \mathcal{C}1 \otimes \ldots \otimes \mathcal{C}n$, and a minimum interest threshold λ. It holds that:*

$$int^{\mathcal{I}}_p(\lambda) \subseteq int^{\mathcal{I}}_f(\lambda)$$

Proof. Consider two real numbers x_1, x_2 in the interval $[0, 1]$. It holds that $x_1 \times x_2 \leq min(x_1, x_2)$. Therefore, for a given pattern i, if in the probabilistic semiring $\otimes\mathcal{C}(i) \geq_p \lambda$, then also in the fuzzy semiring $\otimes\mathcal{C}(i) \geq_f \lambda$.

	$\langle supp, \mathcal{D}, \geq, t, \alpha\rangle$			$\langle avg, weight, \leq, t, \alpha\rangle$		$\langle sum, price, \geq, t, \alpha\rangle$	
	$\mathcal{D}$	t	α	t	α	t	α
$\mathcal{Q}_1$	RETAIL	20	0.8	10000	0.5	20000	0.5
$\mathcal{Q}_2$	RETAIL	20	0.5	10000	0.5	20000	0.5
$\mathcal{Q}_3$	RETAIL	20	0.2	10000	0.5	20000	0.5
$\mathcal{Q}_4$	RETAIL	20	0.8	5000	0.2	20000	0.5
$\mathcal{Q}_5$	RETAIL	20	0.8	5000	0.8	20000	0.5
$\mathcal{Q}_6$	T40I10D100K	800	0.75	15000	0.2	100000	0.5
$\mathcal{Q}_7$	T40I10D100K	800	0.75	15000	0.9	100000	0.5
$\mathcal{Q}_8$	T40I10D100K	800	0.25	15000	0.2	100000	0.2

Fig. 2. Description of queries experimented.



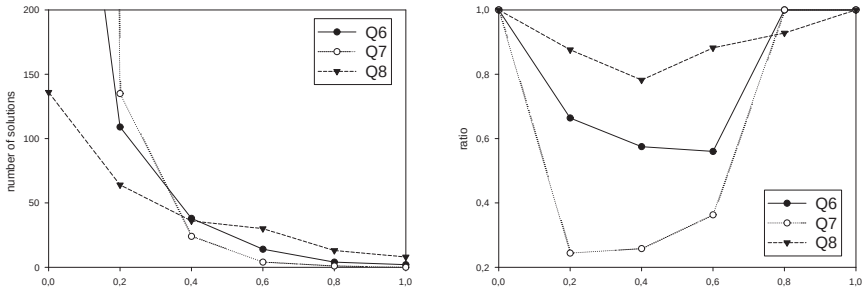
(a) (b)

Fig. 3. Experimental results on the RETAIL dataset with λ ranging in $]0, 1]$ in the probabilistic semiring: number of solutions (a), and ratio with the number of solutions in the fuzzy semiring (b).

When dealing with the probabilistic semiring, we translate the given query to a crisp one. But afterwards, we need a post-processing step in which we select, among the solutions to the crisp query, the λ-interesting patterns. It is natural to ask ourselves how much selective is this post-processing. This could provide a measure of the kind of improvement that one could get by studying and developing ad-hoc techniques, to push probabilistic soft constraints into the pattern extraction computation.

In Figure 3, for the RETAIL dataset and the queries of Figure 2, we report: in (a), the number of λ-interesting patterns in the probabilistic semiring, while in (b) the ratio of this number with the number of solutions in the fuzzy semiring, i.e., $|int_p^{\mathcal{I}}(\lambda)| \,/\, |int_f^{\mathcal{I}}(\lambda)|$. The execution time of the post-processing is not reported in the plots, because in all the experiments conducted, it was always in the order of few milliseconds, thus negligible w.r.t. the mining time. Observing the ratio we can note that it is always equals to 1 for $\lambda = 0$ and $\lambda = 1$. In fact a pattern having at least a constraint for which it returns 0, will receive a semiring value of 0 in both the fuzzy semiring (min combination operator), and the probabilistic semiring ($\times$ combination operator). Similarly, for $\lambda = 1$, to be a solution a pattern must return a value of 1 for all the constraints in the combination, in both the semirings. Then we can observe that this ratio is quite high, always larger than 0.7 in the RETAIL dataset. This is no longer true for the queries on the T40I10D100K dataset, reported in Figure 4 (a) and (b): the ratio reach a minimum value of 0.244 for query $\mathcal{Q}_7$ when $\lambda = 0.2$.

What we can observe is that the ratio does not depend neither on the number of solutions nor on λ (apart the extreme cases 0 and 1). The ratio depends on the softness of the query: the softer the query the lower the ratio, i.e., more patterns discarded by the post-processing. This can be observed in both Figure 3(b) and 4(b): for instance, among the first three queries $\mathcal{Q}_1$ is softer than $\mathcal{Q}_2$ which in turns is softer than $\mathcal{Q}_3$, and this is reflected in the ratio which is lower for $\mathcal{Q}_1$; similarly $\mathcal{Q}_5$ is softer than $\mathcal{Q}_4$ and its ratio is lower; in 4(b) $\mathcal{Q}_8$ is the least soft while $\mathcal{Q}_7$ is the most soft, and accordingly behaves the ratio.



(a) (b)

Fig. 4. Experimental results on the T40I10D100K dataset with λ ranging in $]0, 1]$ in the probabilistic semiring: number of solutions (a), and ratio with the number of solutions in the fuzzy semiring (b).

4 Mining $int_w^{\mathcal{I}}(\lambda)$ (λ-interesting Itemsets on the Weighted Semiring)

While in the fuzzy semiring each pattern has an associated level of preference (or interestingness) for each constraint, and in the probabilistic semiring a value which represents a probability, in the weighted semiring they have an associated cost. Therefore, in the weighted semiring the cost function is defined by summing up the costs of all constraints. According to the informal description given above, the weighted semiring is $S_W = \langle \mathbb{R}^+, min, sum, +\infty, 0 \rangle$.

Example 4. Consider the following weighted instance for the constraints in the query $\mathcal{Q}$ (graphically represented in Figure 5):

- $C_1(supp_{\mathcal{D}}(X)) = \begin{cases} 1750 - supp_{\mathcal{D}}(X), & \text{if } supp_{\mathcal{D}}(X) < 1750 \\ 0, & otherwise. \end{cases}$
- $C_2(avg(X.weight)) = 25 * avg(X.weight)$
- $C_3(sum(X.price)) = \begin{cases} 5 * (60 - sum(X.price)), & \text{if } sum(X.price) < 60 \\ 0, & otherwise. \end{cases}$

Note how the soft version of the constraints are defined in the weighted framework: C_1 for instance, since bigger support is better, gives a cost of 0 when the support is greater than 1750 and an increasing cost as the support decreases. Similarly for constraint C_3: we assign a cost 0 when the sum of prices is at least 60, while the cost increases linearly as the sum of prices shrinks. Constraint C_2 instead aims to have an average weight as lower as possible, and thus larger average weight will produce larger (worse) cost. In this situation we got that:

- $p_1 : C_1 \otimes C_2 \otimes C_3(1700, 0.8, 19) = sum(50, 20, 205) = 275$
- $p_2 : C_1 \otimes C_2 \otimes C_3(1550, 4.8, 54) = sum(200, 120, 30) = 350$
- $p_3 : C_1 \otimes C_2 \otimes C_3(1550, 2.2, 26) = sum(200, 55, 170) = 425$

Therefore, with this particular instance we got that $p_3 <_{S_W} p_2 <_{S_W} p_1$ (remember that the order $\leq_{S_W}$ correspond to the $\geq$ on real numbers). In other terms, p_1 is the most interesting pattern w.r.t. this constraints instance.

Since in the weighted semiring, the values correspond to costs, instead of looking for patterns with an interest level larger than λ, we seek for patterns with a cost smaller than λ.

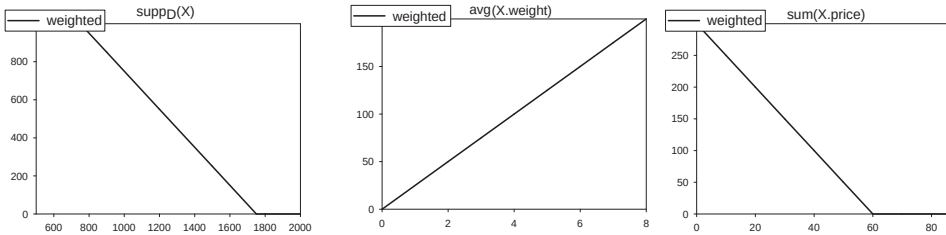
Definition 7. *Let $\mathcal{I} = \{x_1, ..., x_n\}$ be a set of items, where an item is an object with some predefined attributes (e.g., price, type, etc.). A soft constraint on itemsets, based on the weighted semiring, is a function $\mathcal{C} : 2^{\mathcal{I}} \rightarrow \mathbb{R}^+$. Given a combination of such soft constraints $\otimes\mathcal{C} \equiv \mathcal{C}_1 \otimes \ldots \otimes \mathcal{C}_n$, we define the interest level of an itemset $X \in 2^{\mathcal{I}}$ as $\otimes\mathcal{C}(X) = \sum_{i=1,...,n} \mathcal{C}_i(X)$. Given a maximum cost threshold $\lambda \in \mathbb{R}^+$, the λ-interesting itemsets mining problem, requires to compute $int_w^{\mathcal{I}}(\lambda) = \{X \in 2^{\mathcal{I}} | \otimes \mathcal{C}(X) \leq \lambda\}$.*

For sake of simplicity, we restrict to weighted constraints with a linear behavior as those ones described in Figure 5. To describe such simple behavior, we need a new parameter $\beta \in \mathbb{R}^+$ that represents the semiring value associated to the t point (playing the role of the implicitly given 0.5 value for the fuzzy and probabilistic semiring). In other words we provide two points to describe the straight line passing through them: the point (t, β) and the point $(t - \alpha t, 0)$ for $\theta = \leq$ or $(t + \alpha t, 0)$ for $\theta = \geq$. Note that α still plays the role of the softness knob.

Definition 8. *A soft constraint $\mathcal{C}$ on itemsets, based on the weighted semiring, is defined by a sextuple $\langle Agg, Att, \theta, t, \beta, \alpha \rangle$, where: Agg, Att, θ and α are defined as for the fuzzy/probabilistic case (Definition 5), t is a point in the carrier set of the weighted semiring, i.e., $t \in \mathbb{R}^+$, and β represents the semiring value associated to t.*

Example 5. Consider again the query $\mathcal{Q}$ given in Example 1, and its weighted instance graphically described by Figure 5. Such query can be expressed in our constraint language as:

$$\langle supp, \mathcal{D}, \geq, 1500, 250, \frac{1}{6} \rangle, \langle avg, weight, \leq, 5, 125, 1 \rangle, \langle sum, price, \geq, 20, 200, 1 \rangle$$



(C_1) (C_2) (C_3)

Fig. 5. Graphical representation of possible weighted instances of the constraints in in the mining query $\mathcal{Q}$ in Example 1.

For the weighted semiring we can still rely on Proposition 1, which states that a pattern in order to be λ-interesting, must return a semiring value smaller than λ (we are dealing this time with costs; i.e., $\geq_W$ is $\leq$) for each single constraint in the query: this assures us that if a pattern does not satisfy the crisp translation of the given query, it will not be λ-interesting neither in the weighted semiring. In other words we can always use the same methodology described for the probabilistic semiring: translate the query to a crisp one, evaluate it, post-process the result to select the exact solution set.

Definition 9. *Given a weighted soft constraint $\mathcal{C} \equiv \langle Agg, Att, \theta, t, \beta, \alpha\rangle$, and a maximum cost threshold λ, we define the crisp translation of $\mathcal{C}$ w.r.t. λ as:*

$$\mathcal{C}^{\lambda}_{crisp} \equiv \begin{cases} Agg(Att) \leq t - \alpha t + \frac{1}{\beta}\lambda\alpha t, & \textit{if } \theta = \leq \\ Agg(Att) \geq t + \alpha t - \frac{1}{\beta}\lambda\alpha t, & \textit{if } \theta = \geq \end{cases}$$

Example 6. Given the weighted soft constraint $\langle sum, price, \geq, 20, 200, 1\rangle$, its crisp translation is $sum(X.price) \geq 24$ for $\lambda = 180$, it is $sum(X.price) \geq 10$ for $\lambda = 250$.

Proposition 3. *Given the vocabulary of items $\mathcal{I}$, a combination of weighted soft constraints $\otimes\mathcal{C} \equiv \mathcal{C}_1 \otimes \ldots \otimes \mathcal{C}_n$, and a maximum interest threshold λ. Let $\mathcal{C}'$ be the conjunction of crisp constraints obtained by conjoining the crisp translation of each constraint in $\otimes\mathcal{C}$ w.r.t. λ: $\mathcal{C}' \equiv {\mathcal{C}_1}^{\lambda}_{crisp} \wedge \ldots \wedge {\mathcal{C}_n}^{\lambda}_{crisp}$. It holds that:*

$$int^{\mathcal{I}}_{w}(\lambda) \subseteq \{X \in 2^{\mathcal{I}} | \otimes\mathcal{C}(X) \leq \lambda\} = Th(\mathcal{C}')$$

where $Th(\mathcal{C}')$ is the solution set for the crisp problem, according to the notation introduced in Definition 2.

In the following we report the results of some experiments that we have conducted on the same datasets used before for the fuzzy and the probabilistic semirings. We have compared 8 different instances (described in Figure 6) of the query $\mathcal{Q}$:

$$\langle supp, \mathcal{D}, \geq, t, \beta, \alpha\rangle\langle avg, weight, \leq, t, \beta, \alpha\rangle, \langle sum, price, \geq, t, \beta, \alpha\rangle$$

The results of the experiments are reported in Figure 7 and Figure 8. A first observation is that, on the contrary of what happening in the probabilistic and fuzzy semiring,

	$\langle supp, \mathcal{D}, \geq, t, \beta, \alpha\rangle$				$\langle avg, weight, \leq, t, \beta, \alpha\rangle$			$\langle sum, price, \geq, t, \beta, \alpha\rangle$		
	$\mathcal{D}$	t	β	α	t	β	α	t	β	α
$\mathcal{Q}_9$	RETAIL	20	600	0.8	5000	100	0.2	20000	250	0.5
$\mathcal{Q}_{10}$	RETAIL	20	600	0.2	5000	100	0.2	20000	250	0.5
$\mathcal{Q}_{11}$	RETAIL	20	600	0.8	5000	100	0.8	20000	250	0.5
$\mathcal{Q}_{12}$	RETAIL	20	600	0.8	5000	500	0.2	20000	250	0.5
$\mathcal{Q}_{13}$	RETAIL	20	600	0.8	5000	1000	0.2	20000	500	0.5
$\mathcal{Q}_{14}$	T40I10D100K	800	500	0.8	5000	200	0.5	80000	400	0.8
$\mathcal{Q}_{15}$	T40I10D100K	600	600	0.8	15000	500	0.5	80000	400	0.8
$\mathcal{Q}_{16}$	T40I10D100K	1000	500	0.5	15000	500	0.5	100000	600	0.9

Fig. 6. Description of queries experimented.

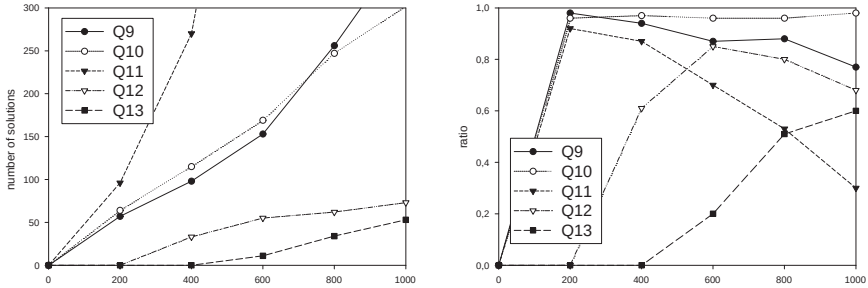


Fig. 7. Experimental results on the RETAIL dataset with λ ranging in $[0, 1000]$ in the weighted semiring: number of solutions (a), and ratio with the number of solutions of the crisp translation (b).

here the larger is λ the larger is the number of solutions. This is trivially because the order of the weighted semiring says that smaller is better. In Figure 7(a) we can observe that queries $\mathcal{Q}_{12}$ and $\mathcal{Q}_{13}$ always return a small number of solutions: this is due to the high values of β in the constraints, which means high costs, making difficult for patterns to produce a total cost smaller than λ. In Figure 7(b) and Figure 8(b) we report the ratio of the number of solution with the cardinality of the theory corresponding to the crisp translation of the queries, i.e., $|int_w^{\mathcal{I}}(\lambda)| \,/\, |Th(\mathcal{C}')|$. This gives a measure of how good is the approximation of the crisp translation, or in other terms, the amount of post-processing needed (which, however, has negligible computational cost). The approximation we obtain using our crisp solver is still quite good but, as we expected, not as good as in the probabilistic semiring. Also in this case, the softer the query the lower the ratio, i.e., the crisp approximation is better for harder constraints (closer to crisp). For instance in Figure 7(b) we can observe that $\mathcal{Q}_{10}$, which is the query with smaller values for the softness parameter α, always present a very high ratio.

5 Mining top-*k* Itemsets

For sake of completeness, in this section we sketch a simple methodology to deal with *top-k* queries, according to [5]. In the following we do not distinguish between the possible semiring instances, we describe the general methodology and leave to the reader to instantiate it to the various semirings.

The main difficult to solve *top-k* queries is that we can know the number of solutions only after the evaluation of a query. Therefore, given k, the simple idea is to repeatedly run λ-interesting queries with different λ thresholds: we start from extremely selective λ (fast mining) decreasing in selectivity, until we do not extract a solution set which is large enough (more than k).

Considering for instance the fuzzy semiring, where the best semiring value is 1: we could start by performing a 0.95-interesting query, and if the query results in a solution set of cardinality larger than k, then we sort the solution according to their semiring

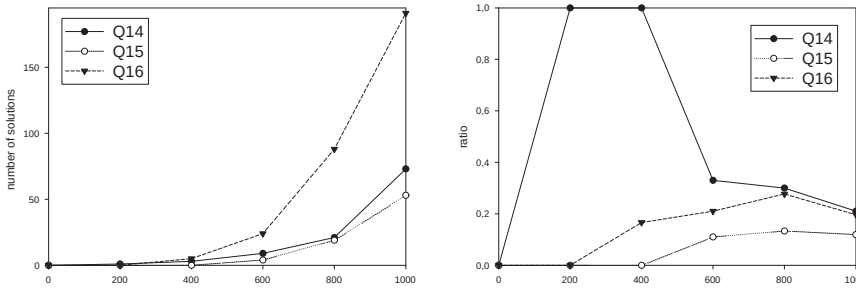


Fig. 8. Experimental results on the T40I10D100K dataset with λ ranging in $[0, 1000]$ in the weighted semiring: number of solutions (a), and ratio with the number of solutions of the crisp translation (b).

value and return the best k, otherwise we slowly decrease the threshold, for instance $\lambda = 0.9$, and so on. Notice that is important to start from a very high threshold in order to perform fast mining extractions with small solution sets, and only if needed decrease the threshold to get more solutions at the cost of longer computations.

6 Related and Future Work

To the best of our knowledge only few works [10, 2] have studied the constraint-based paradigm by a methodological point of view, mainly criticizing some of its weak points. To overcome these weak points in this paper we have proposed the use of soft constraints. A similar approach, based on relaxation of constraints, has been adopted in [1] but for sequential patterns. In the context of sequential patterns, constraints are usually defined by means of regular languages: a pattern is a solution to the query only if it is frequent and it is accepted by the regular language. In this case, constraint-based techniques adopt a deterministic finite automaton to define the regular language.

The use of regular languages transforms the pattern mining process into the verification of which of the sequences of the language are frequent, completely blocking the discovery of novel patterns. In [1] the authors propose a new mining methodology based on the use of constraint relaxations, which assumes that the user is responsible for choosing the strength of the restriction used to constrain the mining process. A hierarchy of constraint relaxations is developed.

We are actually working at the tight integration of the proposed framework over the CONQUEST [7] system: this requires to define methodologies of interaction between the user and the system, e.g., how to define by means of a graphical paradigm the behavior of the soft constraints. We plan also to apply the framework on real-world biomedical problems, where the physicians want to drive the discovery process using their background domain knowledge, but at the same time, hope to discover some novel, unknown, surprising patterns.

References

1. C. Antunes and A.L. Oliveira. Constraint relaxations for discovering unknown sequential patterns. In *Proceedings of the Third International Workshop on Knowledge Discovery in Inductive Databases*, pages 11–32, 2004.
2. R.J. Bayardo. The hows, whys, and whens of constraints in itemset and rule discovery. In *Constraint-Based Mining and Inductive Databases, European Workshop on Inductive Databases and Constraint Based Mining*, pages 1–13, 2004.
3. J. Besson, C. Robardet, J.F. Boulicaut, and S. Rome. Constraint-based concept mining and its application to microarray data analysis. *Intelligent Data Analysis journal*, pages 59–82, 2005.
4. S. Bistarelli and F. Bonchi. Interestingness is not a dichotomy: Introducing softness in constrained pattern mining. In *Proceedings of the Ninth European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 22–33, 2005.
5. S. Bistarelli, P. Codognet, and F. Rossi. Abstracting soft constraints: Framework, properties, examples. *Artificial Intelligence*, (139):175–211, July 2002.
6. S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based Constraint Solving and Optimization. *Journal of the ACM*, 44(2):201–236, Mar 1997.
7. F. Bonchi, F. Giannotti, C. Lucchese, S. Orlando, R. Perego, and R. Trasarti. CONQUEST: a constraint-based querying system for exploratory pattern discovery. In *Proceedings of The 22nd IEEE International Conference on Data Engineering*, pages 22–33, 2006.
8. Ordonez C. et al. Mining constrained association rules to predict heart disease. In *Proceedings of the First IEEE International Conference on Data Mining*, pages 433–440, 2001.
9. H. Fargier and J. Lang. Uncertainty in constraint satisfaction problems: a probabilistic approach. In *Proc. European Conference on Symbolic and Qualitative Approaches to Reasoning and Uncertainty (ECSQARU)*, volume 747 of *LNCS*, pages 97–104. Springer-Verlag, 1993.
10. J. Hipp and H. Güntzer. Is pushing constraints deeply into the mining algorithms really what we want?: an alternative approach for association rule mining. *SIGKDD Explorations*, 4(1):50–55, 2002.
11. A. Lau, SS. Ong, A. Mahidadia, AG. Hoffmann, J. Westbrook, and T. Zrimec. Mining patterns of dyspepsia symptoms across time points using constraint association rules. In *Advances in Knowledge Discovery and Data Mining, Proceedings of the 7th Pacific-Asia Conference*, pages 124–135, 2003.

On Interactive Pattern Mining from Relational Databases

Francesco Bonchi[1], Fosca Giannotti[1], Claudio Lucchese[23], Salvatore Orlando[23], Raffaele Perego[3], and Roberto Trasarti[1]

[1] Pisa KDD Laboratory, ISTI - CNR,
Area della Ricerca di Pisa, Via Giuseppe Moruzzi 1, Pisa, Italy
[2] Computer Science Dep., University Ca' Foscari
Via Torino 155, Venezia Mestre, Italy
[3] Pisa HPC Laboratory, ISTI - CNR,
Area della Ricerca di Pisa, Via Giuseppe Moruzzi 1, Pisa, Italy

Abstract. In this paper we introduce a constraint based querying system devised with the aim of supporting the intrinsically exploratory (i.e., human-guided, interactive, iterative) nature of pattern discovery. Following the Inductive Database vision, our framework provides users with an expressive constraint based query language which allows the discovery process to be effectively driven toward potentially interesting patterns. Such constraints are also exploited to reduce the cost of pattern mining computation. We implemented a comprehensive mining system that can access real world relational databases from which extract data. After a preprocessing step, users mining queries are answered by an efficient pattern mining engine which entails several data and search space reduction techniques. Finally, results are presented to the user, and then stored in the database. New user-defined constraints can be easily added to the system in order to target the particular application considered.

1 Introduction

According to the Inductive Database vision [10], the task of extracting useful and interesting knowledge from data is just an *exploratory* querying process, i.e., human-guided, iterative and interactive. The analyst, exploiting an expressive query language, drives the discovery process through a sequence of complex mining queries, extracts patterns satisfying some user-defined constraints, refines the queries, materializes the extracted patterns as first-class citizens in the database, combines the patterns to produce more complex knowledge, and cross-over the data and the patterns.

Therefore, an Inductive Database system should provide the following features:

Coupling with a DBMS. The analyst must be able to retrieve the portion of interesting data (for instance, by means of SQL queries). Moreover, extracted patterns should also be stored in the DBMS in order to be further queried or mined (*closure principle*).

Expressiveness of the query language. The analyst must be able to interact with the pattern discovery system by specify declaratively how the desired patterns should look like and which conditions they should satisfy. The analyst is supposed to have a high-level vision of the pattern discovery system, without worrying about the details of the computational engine, in the same way as a database user has not to worry about query optimization. The task of composing all constraints and producing the most efficient mining strategy (execution plan) for a given query should be thus completely demanded to the underlying system.

Efficiency of the mining engine. Keeping query response time as small as possible is, on the one hand necessary, since our goal is to give frequent feedbacks to the user allowing realistic human-guided exploration. On the other hand, it is a very challenging task, due to the exponential complexity of pattern discovery computations. To this end, data and search space reduction properties of constraints should be effectively exploited by pushing them within the mining algorithms. Pattern discovery is usually a highly iterative task: a mining session is usually made up of a series of queries (exploration), where each new query adjusts, refines or combines the results of some previous queries. It is important that the mining engine adopts techniques for incremental mining; i.e. reusing results of previous queries, in order to give a faster response to the last query presented to the system, instead of performing again the mining from scratch.

Graphical user interface. The exploratory nature of pattern discovery imposes to the system not only to return frequent feedbacks to the user, but also to provide pattern visualization and navigation tools. These tools should help the user in visualizing the continuous feedbacks form the systems, allowing an easier and human-based identification of the fragments of interesting knowledge. Such tools should also play the role of graphical querying interface: the action of browsing pattern visualization should be tightly integrated (both by a conceptual and engineering point of view) with the action of iteratively querying.

Starting from the above requirements we designed ConQueSt, an exploratory pattern discovery system equipped with a simple, yet powerful, query language (named *SPQL*) and a user friendly interface for accessing the underlying DBMS. ConQueSt is the result of a joint work of the Pisa KDD (*Knowledge Discovery and Delivery*) and HPC (*High Performance Computing*) Laboratories: it is built around a scalable and high-performance constraint-based mining engine exploiting state-of-the-art constraint pushing techniques, as those ones developed in the last three years by our two labs [14, 4, 3, 6, 7].

The constraint-based pattern mining paradigm has been recognized as one of the fundamental techniques for Inductive Databases: user-defined constraints drive the mining process towards potentially interesting patterns only. Moreover, they can be pushed deep inside the mining algorithm in order to deal with the exponential search space curse, achieving better performance [17, 13, 9]. Constraint-based frequent pattern mining has been studied a lot as a query optimization

problem, i.e., developing efficient, sound and complete evaluation strategies for constraint-based mining queries. To this aim, properties of constraints have been studied comprehensively, e.g., *anti-monotonicity*, *succinctness* [13, 12], *monotonicity* [11, 8, 3], *convertibility* [15], *loose anti-monotonicity* [5], and on the basis of such properties efficient computational strategies have been defined.

Definition 1 (Constrained Frequent Itemset Mining). *Let $\mathcal{I} = \{x_1, ..., x_n\}$ be a set of distinct items, where an item is an object with some predefined attributes (e.g., price, type, etc.). An itemset X is a non-empty subset of $\mathcal{I}$. A transaction database $\mathcal{D}$ is a bag of itemsets $t \in 2^{\mathcal{I}}$, usually called transactions. A constraint on itemsets is a function $\mathcal{C} : 2^{\mathcal{I}} \rightarrow \{true, false\}$. We say that an itemset I satisfies a constraint if and only if $\mathcal{C}(I) = true$. We define the theory of a constraint as the set of itemsets which satisfy the constraint: $Th(\mathcal{C}) = \{X \in 2^{\mathcal{I}} \mid \mathcal{C}(X)\}$. The support of an itemset X in database $\mathcal{D}$, denoted $supp_{\mathcal{D}}(X)$, is the number of transactions which are superset of X. Given a user-defined minimum support, denoted σ, an itemset X is called frequent in $\mathcal{D}$ if $supp_{\mathcal{D}}(X) \geq \sigma$. This defines the minimum frequency constraint: $\mathcal{C}_{freq[\mathcal{D},\sigma]}(X) \Leftrightarrow supp_{\mathcal{D}}(X) \geq \sigma$. In general, given a conjunction of constraints $\mathcal{C}$ the constrained frequent itemsets mining problem requires to compute $Th(\mathcal{C}_{freq}) \cap Th(\mathcal{C})$.*

While developing ConQueSt we have tried to reduce as much as possible the gap existing between real-world data stored in relational DBMS, and the constraint-based pattern discovery paradigm, as defined above.

In fact, the data is usually stored in relational databases, and thus in the form of relations and not of transactions. In Section 4 we explain how transactions are defined and constructed both at the query language and at the system level.

Moreover, the constraint-based mining paradigm assumes that the constraints are defined on attributes of the items that are in functional dependency with the items. This rarely the case in real-world data: just think about the price of one item in a market over a period of one month. ConQueSt provides support to solve this cases, allowing the user to reconstruct the ideal situation of functional dependency, for instance, by choosing to take the average of prices of the item in the period as price attribute.

2 Architecture of the system

The ConQueSt architecture, as shown in Fig. 1, is composed of three main modules:

- the Graphical User Interface (GUI);
- the Query Interpreter and and Pre-processor (QIP);
- the Mining Engine (ME).

For portability reasons, the GUI and the QIP have been implemented in Java, while the ME was implemented in C++ in order to provide high performance during the most expensive task.

In our vision, a mining system has to be tightly coupled with DMBS softwares, because databases are the place where data is. Our choice is to allow all the three main components to access independently a database. In fact, the GUI must show to the user the data present in the database, the QIP must retrieve the data of interested and prepare them for the mining engine, which will eventually store the discovered patterns in the database. To this end, the three components stand on a JDBC [1] abstraction layer, in order to provide independency from the particular database server software where data are stored. In fact, the JDBC API provides connectivity to a wide range of SQL databases. ConQueSt for instance, can retrieve data from PostgreSQL as well as from Microsoft Access database servers, and additional compatibility can be provided just by adding the JDBC plug-in provided by the database server software house.

The separation in these modules reflects the separation among different, well defined and independent tasks. In fact, every module could be a single software package running on a different machine. For instance, the GUI may run on the user machine, while the QIP may be located in a different site where a fast access to the database is provided, and finally the ME may be running on an high performance cluster serving many users with many different tasks. Finally, the JDBC layer allow us to ignore the physical location of the database server.

Actually, a communication protocol is established, flowing from the GUI, through the QIP and ending at the mining engine. The GUI, interactively with the user, creates a SPQL query which is then sent to the query interpreter. The latter preprocesses the data of interest and translates part of the SPQL query in an list of constraints. These constraints, and a filesystem pointer to the preprocessed data are finally sent to the ME which can now start the mining process. As long as this simple protocol is fulfilled, every single component can increase its features and improve its functionalities independently from the others.

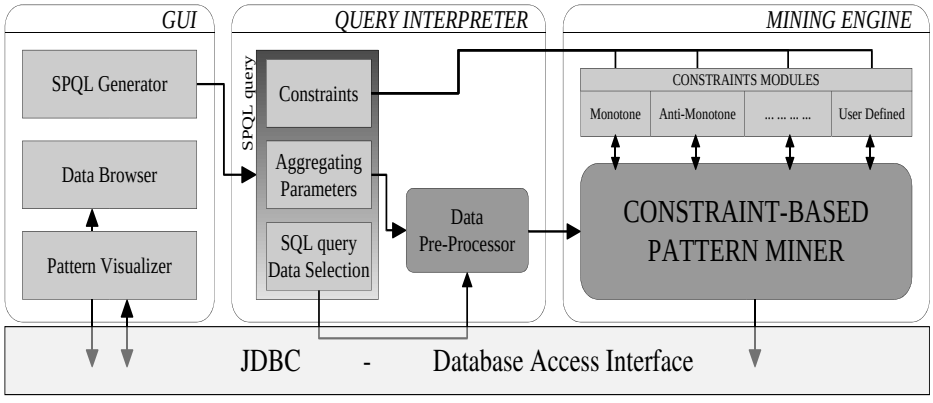


Fig. 1. ConQueSt architecture.

3 Graphical User Interface

The user interface (see a screen-shot in Figure 2) is designed not only to stand in between the user and the mining engine, but also to stand between the user and the data.

Data is assumed to be in the form of a relational database. As soon as the user connects to the database, a set of information and statistics are collected and presented in many ways. The idea is to provide a simple and high level mean to the user to define the mining task. It is simple, since the user can reach his goal just by using user friendly mouse-clicks. Moreover, many high level information and statistics are provided. Finally, the GUI is complete, meaning that any operation related to the definition of a mining query can be done just by mouse-clicks without the need of editing an SPQL query by hand.

Navigating the Structure of the Database. Most of the GUI is dedicated to the *Desk Area*. Here the tables present in the database are showed in a shape of a graph structure. Each vertex of the graph represents a table, and the user may choose to see or not to see all the fields of the table. Each edge of the graph corresponds to a logical link between a foreign key and a primary key. Finally, a *Tables List* helps the user to select, to hide or unhide any of the tables. This gives the user an high level view of the database, allowing him grasp all the relations and connections at a glance, and to focus on the portion of data of interest.

Table-Fields Information and Statistics. Every table maybe actually visualized in the *Table Visualization* panel, but aggregated information are more useful to the user. For this reason ConQueSt shows the data type of each field, statistics of the selected field (e.g. average, minimum, maximum) and a bar or pie chart of the distribution of the values of the selected field. This information may help the user in deciding the discretization parameters and the constraints thresholds.

Interactive SPQL query definition. The first part of the SPQL query consists in the mining view definition, i.e. the set of fields defining transactions, items and attributes. The user may set the mining view simply by left-clicking directly on the *Desk Area* the table-fields of interest. Those fields will be highlighted in the *Desk Area* and reported in the *Mining View Definition* panel. Whenever a mining view definition implicitly require relational joins (e.g. transaction ids and item ids are in different table), they are automatically inserted in the final SPQL query. Constraints and respective threshold may also be set by left-clicking on the *Desk Area* or also using the *Constrains* panel. These facilities allows the user the fully define the mining task just by navigating the dataset graph and using mouse clicks.

Advanced SPQL query definition. At any moment, the user can edit by hand the query in the *SPQL Query* panel. Any modification of the query will be reflected in the rest of the GUI, e.g. by updating the *Mining View Definition* panel. The possibility to edit directly the query, rather than using the gui, does not provide any additional expressive power from a mining point

of view. Anyway, since part of the SPQL query is pure SQL, we can allow the user to exploit complex SQL queries and additional constraints that are not part of the mining task, but rather they are part of the data preparation phase. Moreover, any SQL query can be submitted in place of an SPQL query, providing additional control to the analyst. Finally, before executing the mining task, a preview of the data in the transactional format, together with its items and attributes, is provided in the *Mining View* panel.

Pattern Browser. Result of mining queries are shown to the user in a specialized interface, named *Pattern Browser*. The Pattern Browser provides statistics on the query results, and various functionalities for interactive navigation the set of patterns, such as various kinds of sorting (e.g., cardinality times frequency). The pattern browser also shows the SPQL query that generated the patterns, and allows the user to tune the query parameters according to his needs, for instance, strengthening or relaxing some constraints on the fly. In the pattern browser the user can also invoke some post-processing ore require to materialize the extracted patterns in the underlying database. At the moment the unique kind of post-processing implemented is the extraction of association rules from the patterns results set, but we plan to introduce more complex post-processing which uses extracted patterns as basic bricks to build global models as clustering or classifiers. Also the set of extracted association rules can be materialized as relational tables in the underlying database.

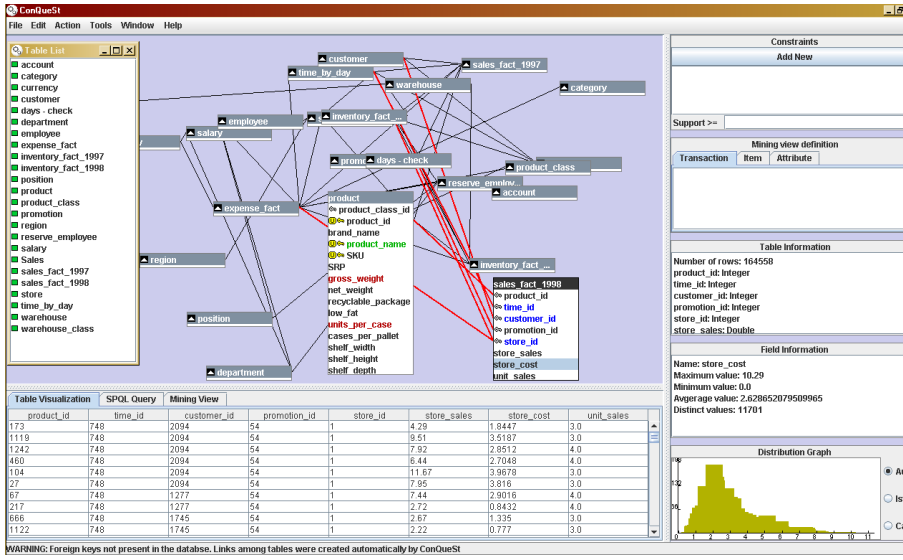


Fig. 2. ConQueSt Graphical User Interface.

4 Query Interpreter and Pre-processor

The second module takes care of interpreting the SPQL query, retrieving from the underlying DBMS the data source, and preparing it for the mining phase.

The preprocessor receives from the GUI a well-formed SQPL query, and it is in charge to accomplish the data preparation step. In fact, the Mining Engine is not able to deal with a relational dataset, it can only read data in a proper format. This format is the one traditionally used in frequent itemsets mining contexts. The input dataset is a collection of transactions, where each transaction is a set of *items*. Each of these items is stored as an integer identifier. In the relational dataset an item may be a string, or even a floating point value, but to feed the mining engine these values have to be discretized or mapped.

Thus, the query interpreter, uses the mining view definition present in the SPQL query to retrieve from the original dataset the data of interest. These data are mapped and translated in a categorical format and finally stored on disk. The rest of the query, i.e. frequency and other constraints, are forwarded to the mining engine together with a pointer the the transactional dataset.

```
1. MINE PATTERNS WITH SUPP>= 5 IN
2. SELECT product.product_name, product.gross_weight, sales_fact_1998.time_id,
   sales_fact_1998.customer_id, sales_fact_1998.store_id
3. FROM [product], [sales_fact_1998]
4. WHERE sales_fact_1998.product_id=product.product_id
5. TRANSACTION sales_fact_1998.time_id, sales_fact_1998.customer_id,
   sales_fact_1998.store_id
6. ITEM product.product_name
7. ATTRIBUTE product.gross_weight
8. CONSTRAINED BY Sum(product.gross_weight)<=30
```

Table 1. An example SPQL Mining Query.

A SPQL query consists of four parts (see Table 1):

1. the user-defined minimum support threshold (line 1);
2. the SQL style `SELECT` statement to specify the data source to be extracted from the DBMS (lines 2–4);
3. the mining view definition by means of `TRANSACTIONS`, `ITEMS` and of `ATTRIBUTES` on which constraints are defined (lines 5–7);
4. the constraints the extracted patterns must satisfy (line 8).

Since the data source is in relational form, a pre-processing step is needed to create a set of transactions, which are the input of any frequent pattern mining system. Transaction are created by grouping `ITEMS` by the attributes specified in the `TRANSACTIONS` clause. For instance, in the query Table 1, transactions are built groping sales by time id, customer id and store id: this means that we

```
<SpqlQuery> ::= (<SqlQuery>| <MineQuery> | <Discretize>)
<MineQuery> ::=<Header><br><SqlQuery><br><MiningDefinition><br><Constraints>
<Header> ::= MINE PATTERNS WITH SUPP >= <Number>
<MiningDefinition> ::= TRANSACTION <Transaction><br> ITEM
<Item><br>[ ATTRIBUTE <Attribute>]
<Transaction> ::= <Field>[<Separator><Transaction>]
<Item> ::= <Field>[<Separator><Item>]
<Attribute> ::= <Field>[<Separator><Attribute>]
<Field> ::= <String>.<String>
<Constraints> ::= CONSTRAINED BY <Function>
<Function> ::= (<FunctionM>(<Field>)<Op><Number> | <FunctionS>(<Field>)<Op><Set> |
               <FunctionN>()<Op><Number> ) [<Separator>(Function)]
<FunctionM> ::= (Minimum | Maximum | Range | Variance | Std_Deviation | Median | Average | Sum)
<FunctionS>::= (Subset_of | Superset_of | Attribute_Subset_of | Attribute_Superset_of )
<FunctionN>::= Length
<Op> ::= (>|<|>=|<=)
<Separator> ::= ,
<br> ::= \n
<Set> ::= <String>[<Separator><Set>]
<Discretize>::= DISCRETIZE <Field> AS <Field> <br> FROM <String> <br> IN
(<Method>| <Intervals>) BINS <br> SMOOTINGH BY <Smethod>
<Method> ::= (EQUALWIDTH | EQUALDEPTH)
<Smethod> ::= (AVERAGE | COUNT | BIN BOUNDARIES)
<Intervals>::= (<Number> <Separator> <Number>)[<Separator> <Intervals>]
<Number> ::= (0-9) [<Number>]
<String> ::= (a-z|A-Z|0-9) [<String>]
```

Table 2. SPQL formal grammar definition. A portion of SPQL language is devoted to the discretization task.

consider a unique transaction when we got the same customer in the same store at the same time. It is worth noting that with this simple mechanism of defining transactions we can easily handle both the inter-attribute and the intra-attribute pattern mining cases.

We have chosen a well defined set of classes of constraints. These constraints have been deeply studied and analyzed in the past few years, in order to find nice properties that can be used at mining time to reduce the computational cost. In particular the ConQueSt system is able to deal with anti-monotone, succinct [13], monotone [3], convertible [16] and loose anti-monotone [7] constraints. Such classes include all the constraints based on the aggregates listed in Table 3.

subset	subset	**supset**	superset
asubset	attributes are subset	**len**	length
asupset	attributes are superset	**acount**	attributes count
min	minimum	**max**	maximum
range	range	**sum**	sum
avg	average	**var**	variance
std	standard deviation	**spv**	sample variance
md	mean deviation	**med**	median

Table 3. The set of available constraints.

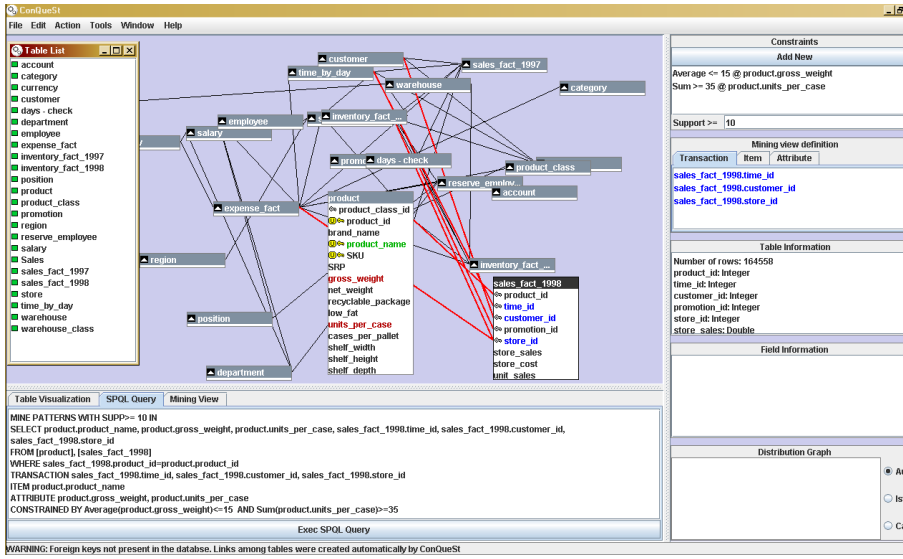


Fig. 3. ConQueSt SPQL Query.

5 The Mining Engine

The last module constitutes the core of the system which mines the transactional dataset and extract the patterns. The mining core guarantees the scalability and performance of the system by exploiting efficient mining algorithms and data reduction techniques coming from the constraint-based mining paradigm.

The ConQueSt mining engine is based on DCI [14], a high performance, Apriori-like, frequent itemsets mining algorithm, with has the nice feature of being resource and data aware. It is resource aware, because, unlike many other algorithms, it performs the first iterations out-of-core, reducing the dataset and rewriting it directly to the disk. When the dataset is small enough, it is converted and stored as a vertical bitmap in main memory. It is data aware because its behavior changes in presence of sparse or dense datasets. It uses a compact representation in the case of sparse datasets, and detects highly correlated items to save bitwise works in the case of dense datasets. ConQueSt by inheriting the same characteristics, is extremely robust and able to effectively mine different kinds of datasets, regardless of their size.

Although the ConQueSt mining engine adopts a level-wise Apriori-like visit of the lattice, thanks to its internal vertical bitwise representation of the dataset, and associated counting strategy, it performs better than other state-of-the-art algorithms that exploit a depth first visit. Moreover, as we have shown in our previous works [3, 7], adopting a level-wise strategy has the great advantage of

allowing the exploitation of different kinds of constraints all together by means of data-reduction. At each iteration of the mining process, the dataset is pruned by exploiting the independent data reductions properties of all user-specified constraints. Our framework, exploits a real synergy of all constraints that the user defines for the pattern extraction: each constraint does not only play its part in reducing the data, but this reduction in turns strengthens the pruning power of the other constraints. Moreover data-reduction induces a pruning of the search space, which in turn strengthens future data reductions. The orthogonality of the exploited constraint pushing techniques has a twofold benefit: first, all the techniques can be amalgamated together achieving a very efficient computation.

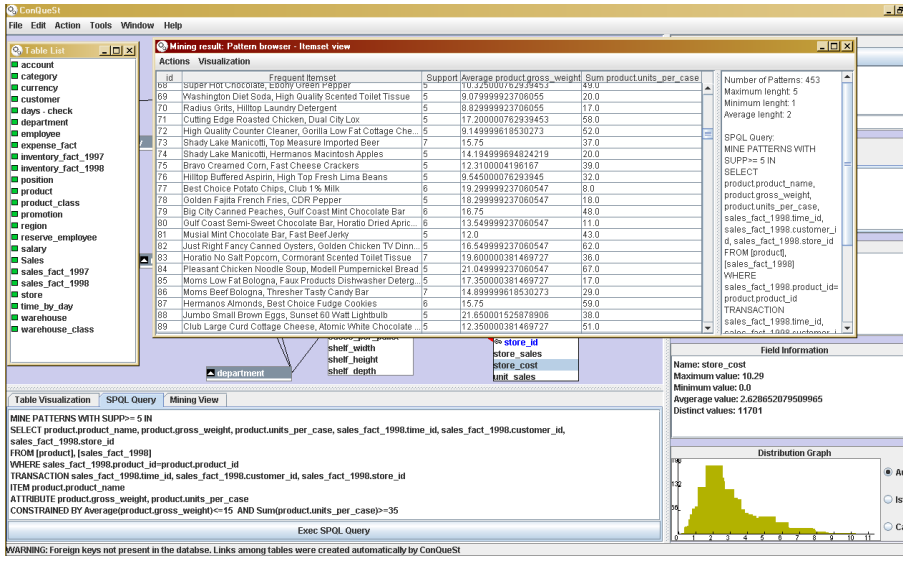


Fig. 4. the output of a mining query in the Pattern Browser.

Moreover, since we have precisely identified classes of constraints which share nice properties, each class has been implemented as an independent module, which plays its role in precise points of the computation. Each module is then instantiated on the basis of the specific constraints supplied by the user. ConQueSt can be easily extended to cope with new user-defined constraints. In fact, it is not the constraint itself that performs data and search space reductions directly, but it is instead the overall framework which exploits constraints classes properties during the computation. Therefore, in order to define a novel constraint, and embed it in the computational framework, it is sufficient to communicate to the system to which classes (possibly more than one) it belongs.

6 Conclusion

Many distinguishing features make ConQueSt a unique system:

Large variety of constraints handled - To the best of our knowledge, ConQueSt is the only system able to deal with so many different constraints all together, and provide the opportunity of easily defining new ones. While some prototypes for constraint-based pattern discovery exist, they are usually focused on few kinds of constraints, and their algorithmic techniques can not be easily extended to other constraints.

Usability - ConQueSt has been devised to fruitfully deal with real-world problems. The user friendly interface, the pre-processing capabilities and the simple connectivity to relational DBMS, make it easy for the user to immediately start to find nuggets of interesting knowledge in her/his data. Modularity and extensibility make the system able to adapt to changing application needs. Efficiency, robustness and scalability make possible to mine real-world huge datasets.

Robustness and resources awareness - One of the main drawbacks of the state-of-the art software for pattern discovery, is that it usually fails to mine large amounts of data due to memory lack. In this sense, ConQueStis robust, since huge datasets are mined out-of-core until the data-reduction framework reduces the dataset size enough to move it in-core.

Efficiency - ConQueSt is a high performance mining software. As an Example consider Figure 5 where we compare execution times of ConQueSt against $\mathcal{FIC}^{\mathcal{A}}$ and $\mathcal{FIC}^{\mathcal{M}}$ [16], two depth first algorithms, ad-hoc devised to deal with the `avg` constraint (the one used in the comparison).

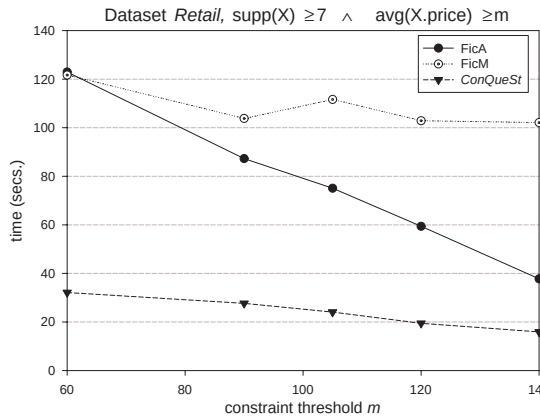


Fig. 5. Performance comparison.

Even tough ConQueSt is already fruitfully usable on real-world problems, many direction must be explored in the next future: efficient incremental min-

ing, soft constraints [2], advanced visualization techniques, more complex post-processing, building global models from the interesting patterns, mining patterns from complex data such as sequences and graphs. We are continuously developing new functionalities of ConQueSt.

References

1. `http://java.sun.com/products/jdbc/`.
2. S. Bistarelli and F. Bonchi. Interestingness is not a dichotomy: Introducing softness in constrained pattern mining. In *Proceedings of the Ninth European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 22–33, 2005.
3. F. Bonchi, F. Giannotti, A. Mazzanti, and D. Pedreschi. ExAMiner: Optimized level-wise frequent pattern mining with monotone constraints. In *Proceedings of ICDM'03*.
4. F. Bonchi, F. Giannotti, A. Mazzanti, and D. Pedreschi. ExAnte: Anticipated data reduction in constrained pattern mining. In *Proceedings of PKDD'03*.
5. F. Bonchi and C. Lucchese. Pushing tougher constraints in frequent pattern mining. In *Proceedings of the Ninth Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'05)*, Hanoi, Vietnam, 2005.
6. Francesco Bonchi and Claudio Lucchese. On closed constrained frequent pattern mining. In *Proceedings of ICDM'04*.
7. Francesco Bonchi and Claudio Lucchese. Pushing tougher constraints in frequent pattern mining. In *Proceedings of The Ninth Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD '05).May 18 - 20, 2005. Hanoi, Vietnam.*
8. Cristian Bucila, Johannes Gehrke, Daniel Kifer, and Walker White. DualMiner: A dual-pruning algorithm for itemsets with constraints. In *Proceedings of ACM SIGKDD'02*.
9. Jiawei Han, Laks V. S. Lakshmanan, and Raymond T. Ng. Constraint-based, multidimensional data mining. *Computer*, 32(8):46–50, 1999.
10. T. Imielinski and H. Mannila. A database perspective on knowledge discovery. *Comm. Of The Acm*, 39:58–64, 1996.
11. Stefan Kramer, Luc De Raedt, and Christoph Helma. Molecular feature mining in hiv data. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining, August 26-29, 2001, San Francisco, CA, USA*, pages 136–143, 2001.
12. Laks V. S. Lakshmanan, Raymond T. Ng, Jiawei Han, and Alex Pang. Optimization of constrained frequent set queries with 2-variable constraints. *SIGMOD Record*, 28(2), 1999.
13. Raymond T. Ng, Laks V. S. Lakshmanan, Jiawei Han, and Alex Pang. Exploratory mining and pruning optimizations of constrained associations rules. In *Proceedings of the ACM SIGMOD'98*.
14. S. Orlando, P. Palmerini, R. Perego, and F. Silvestri. Adaptive and Resource-Aware Mining of Frequent Sets. In *Proc. of the 2002 IEEE Int. Conference on Data Mining (ICDM'02)*, pages 338–345, Maebashi City, Japan, December 2002.
15. Jian Pei and Jiawei Han. Can we push more constraints into frequent pattern mining? In *Proceedings of ACM SIGKDD'00*.
16. Jian Pei, Jiawei Han, and Laks V. S. Lakshmanan. Mining frequent item sets with convertible constraints. In *(Proceedings of ICDE'01)*.
17. Ramakrishnan Srikant, Quoc Vu, and Rakesh Agrawal. Mining association rules with item constraints. In *Proceedings of ACM SIGKDD'97*.

Analysis of Time Series Data with Predictive Clustering Trees

Sašo Džeroski[1], Ivica Slavkov[1], Valentin Gjorgjioski[1], and Jan Struyf[2,3]

[1]Dept. of Knowledge Technologies, Jožef Stefan Institute
Jamova 39, 1000 Ljubljana, Slovenia
[2]Dept. of Computer Science, Katholieke Universiteit Leuven
Celestijnenlaan 200A, 3001 Leuven, Belgium
[3]Dept. of Biostatistics and Medical Informatics, Univ. of Wisconsin, Madison
1300 University Avenue, Madison, WI 53706, USA

Abstract. Predictive clustering is a general framework that unifies clustering and prediction. This paper investigates how to apply this framework to cluster time series data. The resulting system, Clus-TS, constructs predictive clustering trees (PCTs) that partition a given set of time series into homogeneous clusters. In addition, PCTs provide a symbolic description of the clusters. The paper considers several distance metrics to measure cluster homogeneity (both quantitative and qualitative). We evaluate Clus-TS on time series data from microarray experiments. Each data set records the change over time in the expression level of yeast genes in response to a change in environmental conditions. Our evaluation shows that Clus-TS is able to identify interesting clusters of genes with similar responses. Clus-TS is part of a larger project where the goal is to investigate how global models can be combined with inductive databases.

1 Introduction

Predictive clustering is a general framework that combines clustering and prediction [1]. Predictive clustering partitions the data set into a set of clusters such that the instances in a given cluster are similar to each other and dissimilar to the instances in other clusters. In this sense, predictive clustering is identical to regular clustering [7]. The difference is that predictive clustering associates a predictive model to each cluster. This model assigns instances to clusters and provides predictions for new instances. So far, decision trees [1, 17] and rule sets [19] have been used in the context of predictive clustering.

This paper investigates how predictive clustering can be applied to cluster time series [10]. A time series is an ordered sequence of measurements of a continuous variable that changes over time. Fig. 1.a presents an example of eight time series partitioned into three clusters: cluster C_1 contains time series that increase and subsequently decrease, C_2 has mainly decreasing time series and C_3 mainly increasing ones. Fig. 1.b shows a so-called predictive clustering tree (PCT) for this set of clusters. This is the predictive model associated with the clustering.

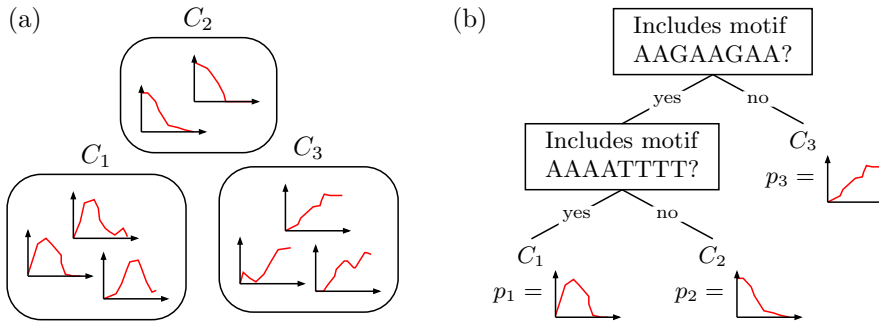


Fig. 1. (a) A set of time series clustered into three clusters. (b) A predictive clustering tree associated with this clustering. Each leaf of the tree corresponds to one cluster.

We propose a new algorithm called Clus-TS (Clustering-Time Series) that constructs trees such as the one shown in Fig. 1.b. Clus-TS instantiates the general PCT induction algorithm proposed by Blockeel et al. [1] to the task of time series clustering. This is non-trivial because the general algorithm requires computing a prototype for each cluster and for most distance metrics suitable for time series clustering, no closed algebraic form prototype is known.

We evaluate Clus-TS on time series data from microarray experiments [5]. Each data set records the change over time in the expression level of yeast genes in response to a change in environmental conditions. Besides the time series, various other data about each gene are available. Here, we consider motifs, which are subsequences that occur in the amino-acid sequence of many genes. The motifs appear in the internal nodes of the PCT (Fig. 1.b) and provide a symbolic description of the clusters. C_1 includes, for example, all genes that have motifs AAGAAGAA and AAAATTTT. This is related to itemset constrained clustering [15], which clusters vectors of numeric values and constrains each cluster by means of an itemset.

So far, most research on inductive databases (IDBs) [6, 3] has focused on local models (i.e., models that apply to only a subset of the examples), such as frequent item sets and association rules. Clus-TS is part of a larger project [4, 17, 19] were the goal is to investigate how IDBs can be extended to global models, such as decision trees and neural networks. Predictive clustering has been argued to provide a general framework unifying clustering and prediction, two of the most basic data mining tasks, and is therefore an excellent starting point for extending IDBs to global models [19]. In particular, we are interested in developing a system that is applicable to clustering and prediction in many application domains, including bioinformatics. Extending PCTs to time series clustering is a step in this direction.

2 Predictive Clustering Trees

2.1 Prediction, Clustering, and Predictive Clustering Trees

Predictive modeling aims at constructing models that can predict a target property of an object from a description of the object. Predictive models are learned from sets of examples, where each example has the form (D, T), with D being an object description and T a target property value.

Clustering [7], on the other hand, is concerned with grouping objects into subsets of objects (called clusters) that are similar w.r.t. their description D. There is no target property defined in clustering tasks. In conventional clustering, the notion of a distance (or conversely, similarity) is crucial: examples are considered to be points in a metric space and clusters are constructed such that examples in the same cluster are close according to a particular distance metric. A prototype (prototypical example) may be used as a representative for a cluster. The prototype is the point with the lowest average (squared) distance to all the examples in the cluster, i.e., the mean or medoid of the examples.

Predictive clustering [1] combines elements from both prediction and clustering. As in clustering, we seek clusters of examples that are similar to each other, but in general taking both the descriptive part and the target property into account (the distance metric is defined on $D \cup T$). In addition, a predictive model must be associated to each cluster. The predictive model assigns new instances to clusters based on their description D and provides a prediction for the target property T. A well-known type of model that can be used to this end is a decision tree [13]. A decision tree that is used for predictive clustering is called a predictive clustering tree (PCT, Fig. 1.b). Each node of a PCT represents a cluster. The conjunction of conditions on the path from the root to that node gives a description of the cluster. Essentially, each cluster has a symbolic description in the form of a rule (IF conjunction of conditions THEN cluster), while the tree structure represents the hierarchy of clusters[1].

In Fig. 1, the description D of a gene consists of the motifs that appear in its sequence and the target property T is the time series recorded for the gene. In general, we could include both D and T in the distance metric. We are, however, most interested in the time series part. Therefore, we define the distance metric only on T. We consider various distance metrics in Section 3.2. The resulting PCT (Fig. 1.b) represents a clustering that is homogeneous w.r.t. T and the nodes of the tree provide a symbolic description of the clusters. Note that, while we are mainly interested in clustering, the tree can also be used for prediction: use the tree to assign a new instance to a leaf and take the prototype time series (denoted with p_i in Fig. 1.b) of the corresponding cluster as prediction.

2.2 Building Predictive Clustering Trees

Fig. 2 presents the generic induction algorithm for PCTs [1]. It is a variant of the standard greedy top-down decision tree induction algorithm [13]. It takes

[1] This idea was first used in conceptual clustering [12].

procedure PCT(I) **returns** tree
1: $(t^*, h^*, \mathcal{P}^*) = \text{BestTest}(I)$
2: **if** $t^* \neq none$ **then**
3: **for each** $I_k \in \mathcal{P}^*$ **do**
4: $tree_k = \text{PCT}(I_k)$
5: **return** node(t^*, $\bigcup_k \{tree_k\}$)
6: **else**
7: **return** leaf(prototype(I))

procedure BestTest(I)
1: $(t^*, h^*, \mathcal{P}^*) = (none, \infty, \emptyset)$
2: **for each** possible test t **do**
3: $\mathcal{P} =$ partition induced by t on I
4: $h = \sum_{I_k \in \mathcal{P}} \frac{|I_k|}{|I|} Var(I_k)$
5: **if** $(h < h^*) \wedge acceptable(t, \mathcal{P})$ **then**
6: $(t^*, h^*, \mathcal{P}^*) = (t, h, \mathcal{P})$
7: **return** $(t^*, h^*, \mathcal{P}^*)$

Fig. 2. The generic PCT induction algorithm.

as input a set of instances I (in our case genes described by motifs and their associated time series). The main loop searches for the best acceptable test (motif) that can be put in a node. If such a test t^* can be found then the algorithm creates a new internal node labeled t^* and calls itself recursively to construct a subtree for each cluster in the partition $\mathcal{P}^*$ induced by t^* on the instances. If no acceptable test can be found, then the algorithm creates a leaf (*acceptable* is the stop criterion of the algorithm, such as maximum tree depth or minimum number of instances).

Up till here, the description is no different from that of a standard decision tree learner. The main difference is the heuristic that is used for selecting the tests. For PCTs, this heuristic minimizes the average variance in the created clusters (weighted by cluster size, see line 4 of BestTest). Minimizing variance maximizes cluster homogeneity. Section 4 discusses how cluster variance can be defined for time series.

3 Clustering Time Series

3.1 Clustering Algorithms

One of the most widely used clustering approaches is hierarchical clustering, due to the great visualization power it offers [8, 11]. Hierarchical clustering produces a nested hierarchy of groups of similar objects, according to a pairwise distance matrix of the objects. One of the advantages of this method is its generality; the user does not need to provide any parameters such as the number of clusters. However, its application is limited to small datasets, due to its quadratic computational complexity. A faster clustering method is k-means [2].

3.2 Distance Measures

Both hierarchical and k-means clustering require that a distance metric is defined on the instances. Also PCTs require a distance metric to define cluster variance (Section 4). The most commonly used distance metrics are the Euclidean and Manhattan distance. These metrics are, however, less appropriate for clustering time series, because they mainly capture the difference in scale and baseline. In time series clustering, we are usually more interested in the difference in shape of

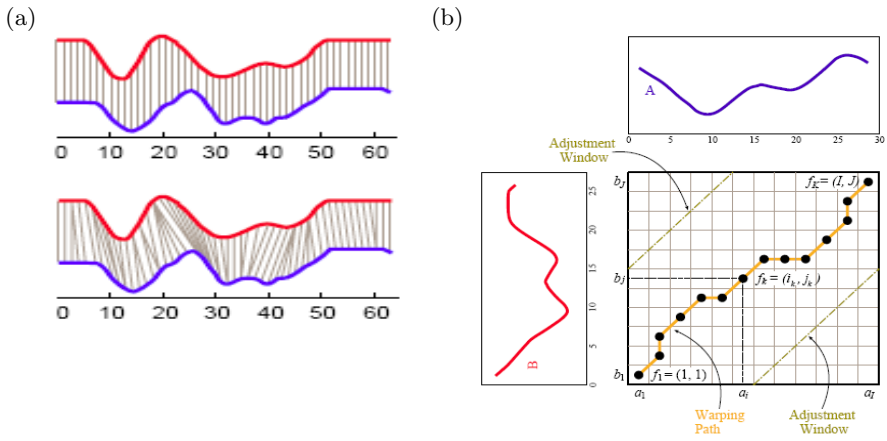


Fig. 3. (a) Euclidean distance (top) compared to DTW (bottom). (b) A warping path. (Artwork courtesy of Eamonn Keogh.)

the time series. Below, we discuss three distance metrics that have been proposed for clustering time series.

Correlation The correlation coefficient $r(X, Y)$ between two time series X and Y is calculated as

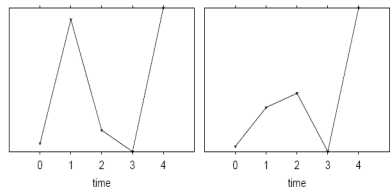
$$r(X,Y) = \frac{E[(X - E[X]) \cdot (Y - E[Y])]}{E[(X - E[X])^2] \cdot E[(Y - E[Y])^2]},$$

where $E(V)$ denotes expectation (i.e., mean value) of V. $r(X, Y)$ measures the degree of linear dependence between X and Y. It has the following intuitive meaning in terms of the shapes of X and Y. r close to -1 means that X and Y have "mirrored" shapes, r close to 0 means that the shapes are unrelated (and consequently dissimilar), and r close to 1 means that the shapes are similar. Following this intuitive interpretation of correlation we can define the distance between two time series as $D_r(X, Y) = \sqrt{0.5 \cdot (1 - r(X, Y))}$. D_r has, however, two drawbacks. First, it is difficult to properly estimate correlation if the number of observations is small (i.e., a short time series). Second, r can only capture the linear dependencies between the time series. Two time series that are non-linearly related will be distant from each other according to this metric.

Dynamic Time Warping Dynamic Time Warping (DTW) [14] can capture non-linear distortion along the time axis. It accomplishes this by assigning multiple values of one of the time series to a single value of the other. As a result, DTW corresponds more to human intuition. Fig. 3.a illustrates DTW and compares it to the Euclidean distance.

$D_{\mathrm{DTW}}(X, Y)$ with $X = \alpha_1, \alpha_2, \ldots, \alpha_I$, $Y = \beta_1, \beta_2, \ldots, \beta_J$ is defined based on the notion of a warping path between X and Y. A warping path is a sequence of grid points $F = f_1, f_2, \ldots, f_K$ on the $I \times J$ plane (Fig. 3.b). Let the distance

(a)



(b)

$Diff(q_1, q_2)$	increase	no-change	decrease
increase	0	0.5	1
no-change	0.5	0	0.5
decrease	1	0.5	0

Fig. 4. (a) Two time series X and Y. $D_r(X, Y) = 0.694$ and $D_{\text{qual}}(X, Y) = 0.2$. (b) The definition of $Diff(q_1, q_2)$.

between two values α_{i_k} and β_{j_k} be $d(f_k) = |\alpha_{i_k} - \beta_{j_k}|$, then an evaluation function $\Delta(F)$ is given by $\Delta(F) = 1/(I + J) \sum_{k=1}^{K} d(f_k) w_k$. The weights w_k are as follows: $w_k = (i_k - i_{k-1}) + (j_k - j_{k-1}), i_0 = j_0 = 0$. The smaller the value of $\Delta(F)$, the more similar X and Y are. In order to prevent excessive distortion, we assume an adjustment window ($|i_k - j_k| \leq r$). $D_{\text{DTW}}(X, Y)$ is the minimum of $\Delta(F)$. D_{DTW} can be computed with dynamic programming in time $O(IJ)$.

A Qualitative Distance A third distance metric is the qualitative metric proposed by Todorovski et al. [18]. It is based on a qualitative comparison of the shape of the time series. Consider two time series X and Y (Fig. 4.a). We choose a pair of time points i and j and we observe the qualitative change of the value of X and Y at these points. There are three possibilities: increase ($X_i > X_j$), no-change ($X_i \approx X_j$), and decrease ($X_i < X_j$). D_{qual} is obtained by summing the difference in qualitative change observed for X and Y for all pairs of time points, i.e.,

$$D_{\text{qual}}(X, Y) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2 \cdot Diff(q(X_i, X_j), q(Y_i, Y_j))}{N \cdot (N - 1)}$$

with $Diff(q_1, q_2)$ a function that defines the difference between different qualitative changes (Fig. 4.b). Roughly speaking, D_{qual} counts the number of disagreements in change of X and Y. D_{qual} does not have the drawbacks of the correlation based measure. First, it can be computed for very short time series, without decreasing the quality of the estimate. Second, it captures the similarity of shape of the time series, regardless of whether their dependence is linear or non-linear.

4 PCTs for Time Series Clustering

4.1 Computing Cluster Variance

Recall from Section 2.2 that the PCT algorithm requires a measure of cluster variance in its heuristics. The variance of a cluster C can be defined based on a distance metric as $Var(C) = \frac{1}{|C|} \sum_{X \in C} d^2(X, p)$, with p the prototype of C. To cluster time series, d should be a distance metric defined on time series, such as

the ones discussed in the previous section. Next, one should decide on a representation for the prototype. We consider three possibilities: (1) the prototype is a probability distribution over time series, (2) the prototype is an arbitrary time series, and (3) the prototype is one of the time series from the cluster. Note that the first representation is the most expressive – it captures most information about the cluster. Once we have decided on a representation, the prototype can be computed as $p = \text{argmin}_q \sum_{X \in C} d^2(X, q)$. In representation (3), the prototype can be computed with $|C|^2$ distance computations by trying for q all time series in the cluster. In the other representations, the space of candidate prototypes is infinite. This means that either a closed algebraic form for the prototype is required or that one should resort to approximative algorithms to compute the prototype. To our knowledge, no closed form prototype is known for any of the three distance metrics discussed in Section 3.2.

An alternative way to define cluster variance is based on the sum of the squared pairwise distances (SSPD) between the cluster elements, i.e., $Var(C) = \frac{1}{|C|^2} \sum_{X \in C} \sum_{Y \in C} d^2(X, Y)$. The advantage of this approach is that no prototype is required. It also requires $|C|^2$ distance computations. This is the same time complexity as the approach with the prototype in representation (3). Hence, using the definition based on a prototype is only more efficient if the prototype can be computed in time linear in the cluster size. This is the case for the Euclidean distance in combination with as prototype the pointwise average of the time series. For the other distance metrics, no such prototypes are known. Therefore, we choose to estimate cluster variance using the SSPD.

4.2 Cluster Prototypes for the Tree Leaves

The PCT algorithm places cluster prototypes in its leaves. Note that these are only included to be inspected by the end user and do not influence the outcome of the algorithm in any other way. For these prototypes, we use representation (3) as discussed above.

5 Analyzing Short Time Series Gene Expression Data

5.1 The Problem

We consider DNA microarray analysis as an interesting area of application for clustering short time series. When working with microarrays sample size is always an issue; the time series have around 4-10 observations. Clustering genes by their time expression pattern makes sense because genes, which are co-regulated or have a similar function, under certain conditions, will have a similar temporal profile. Instead of simply clustering the expression time series, and later on elucidating the characteristics of the obtained clusters, we perform constrained clustering with PCTs. We implement the method for computing cluster variance based on the time series distance metrics into the PCT induction system Clus[2]

[2] http://www.cs.kuleuven.be/~dtai/clus/.

[17]; we call the resulting system Clus-TS. The advantage of PCTs is that they can construct clusters by employing already known features as their descriptors.

5.2 DNA Microarray Time Series

The data that we use for our experiments is from a previous study conducted by Gasch et al. [5]. The purpose of the study is to explore the changes in gene expression levels of yeast (Saccharomyces cerevisiae) under diverse environmental stresses. Various sudden changes in the environmental conditions are tested, ranging from heat shock to amino acid starvation for a prolonged period of time. Microarrays are used to measure the gene expression levels of around 5000 genes. Samples are collected at different time points and different number of times, depending on the particular environmental conditions. The data is log-transformed according to a time-zero point of yeast cells under normal environmental conditions.

5.3 Frequent DNA Subsequences as Features

We also obtain DNA sequences (ORFs – open reading frames) of yeast genes from the Stanford database. In order to connect the DNA sequences of yeast with their corresponding time expression profiles, we mine frequent subsequences (motifs) with FAVST [9] and use these as features. We select constraints that generate a reasonable number of motifs. Basically there are three parameters that we can use as constraints: the frequency of the motif, and the minimum and maximum motif length. As frequency we consider the number of genes in which a motif appears, rather than the absolute number of occurrences of the particular motif. We set the frequency threshold to 25%. The minimum motif length is 8 nucleotides. No limit was set for the maximum motif length. The longest motif that satisfies the frequency constraint has 10 nucleotides. We obtain approximately 300 frequent motifs and use these as features in the PCTs.

5.4 Results

We perform constrained clustering of the short time series by using the frequent motifs as constraints. We construct PCTs with Clus-TS for three datasets: Amino Acid Starvation, Diauxic Shift and Heat Shock. We set the maximum tree depth stop criterion to 3. A sample of the output is presented in Fig. 5. It can be interpreted as: if gene X has/does not have sequence_1, sequence_2... sequence_N then its time expression profile will have a signature as predicted by the prototype in the corresponding leaf.

Amino Acid Starvation The cluster prototypes obtained for the first data set, where yeast is exposed to amino acid starvation, are shown in Fig. 6. These clusters are obtained by using the qualitative distance measure. For the genes in clusters 1-4 we can see that through time, their functioning is constantly repressed. This is somewhat consistent with the expected result of constant starvation, which is reduced cellular activity and growth. The genes in clusters 5-8

```
AGAAAAAA = 0
+--yes: AAAAGAAA = 0
|       +--yes: TTTTTTTC = 0
|       |       +--yes: [0.74,0,-0.2,-0.17,-0.03]: 670;
|       |       +--no:  [0.98,0.14,0.23,0.1,-0.37]: 717;
|       +--no:  AAAAAAAG = 0
|               +--yes: [0.35,0.54,-0.29,-0.52,-0.73]: 696;
|               +--no:  [0.49,0.47,0.33,0.08,0.23]: 659;
+--no: [0.29,-0.3,-0.31,-0.12,-0.21]: 665;
```

Fig. 5. Sample of the output of the Clus-TS system.

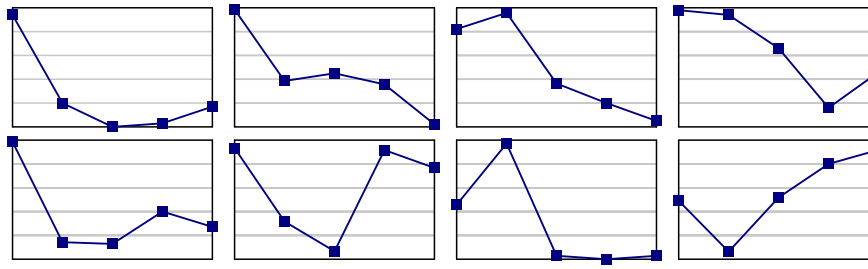


Fig. 6. Plot of cluster prototypes for the Amino Acid Starvation experiment.

show an initial down-regulation and then slowly regress towards their previous state.

Heat Shock Clusters 5 and 7 (Fig. 7) have a distinct expression profile that can be interpreted in the sense of the study by Gasch et al. The heat shock elicits rapid up-regulation of one group of genes and almost symmetrical down-regulation of another group. This is only a transitional state after which the cell adapts to the new condition and then the expression levels of the up/down regulated genes slowly regress to their normal levels.

Diauxic Shift The yeast cells exhibit a similar, although a less coherent response compared to starvation conditions. Two large clusters (clusters 1 and 3, Fig. 8), are down-regulated exhibiting a similarity with starvation, but all

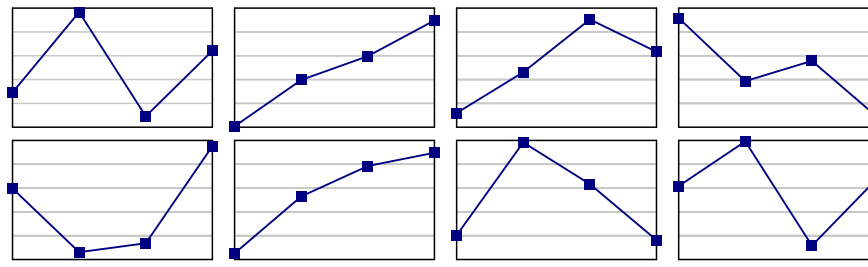


Fig. 7. Plot of cluster prototypes for the Heat Shock experiment.

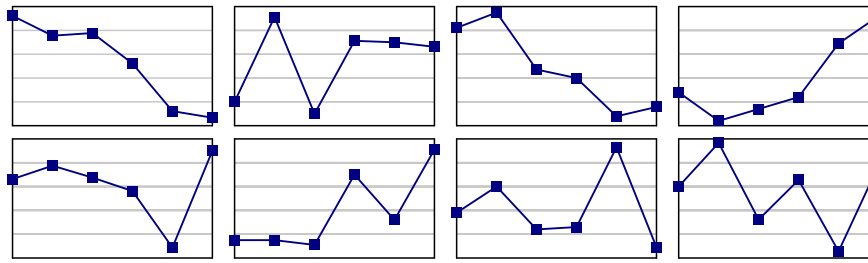


Fig. 8. Plot of cluster prototypes for the Diauxic Shift experiment.

other clusters (except the up-regulation of genes in cluster 4) do not show a coherent response.

6 Future Work

A drawback of computing cluster variance based on the SSPD is that its computational complexity is quadratic in the cluster size. As a result, the current approach does not scale well to large data sets. In future work, we plan to try methods to approximate cluster variance. For example, one could compute variance using the pointwise average of the time series (the exact prototype for the Euclidean distance) as approximate prototype for the other distance metrics. This would allow one to compute an approximate measure of cluster variance in $O(|C|)$ time. Another way of accomplishing the same is to sample $|C|$ pairs of time series from the cluster and approximate the SSPD based only on these pairs. In future work, we plan to assess how these and other approximations compare to the exact definitions of variance.

It would furthermore be interesting to extend the experimental evaluation. This includes testing more datasets (both microarray and other types of short time series data), working with domain experts to interpret the clusterings, and using more types of descriptive attributes (e.g., gene ontology terms). The knowledge gained form these experiments can potentially be used to annotate genes of currently unknown function.

We plan to incorporate different types of constraints in our models. This is important in the context of inductive databases because the inductive queries might include various types of constraints on the resulting PCTs. Our current system already includes accuracy and size constraints [17]. In further work we wish to investigate constraints more specific to clustering and in particular clustering of time series.

Another direction of research is investigating how PCTs and in particular PCTs for clustering time series can be integrated tightly with inductive databases. Fromont and Blockeel [4] and Slavkov et al. [16] present ongoing work in this direction.

7 Conclusion

This paper provides a proof-of-concept that predictive clustering trees (PCTs) can be used to analyze time series data. The main advantage of using PCTs over other clustering algorithms, such as hierarchical clustering and k-means, is that PCTs cluster the time series and provide a description of the clusters at the same time. This allows one to relate various heterogeneous data types and to draw conclusions about their relations, as we have shown in our experimental evaluation.

Using PCTs for time series data is non-trivial because for many appropriate distance metrics (correlation based, dynamic time warping, and a qualitative metric), no closed algebraic form for the prototype is known. Therefore, we propose to compute cluster variance based on the sum of pairwise distances between the cluster elements. This method has not been used previously in predictive clustering and is one of the contributions of the paper.

Our approach combines local models (motifs of DNA) with global models (PCTs). The local models are used to describe clusters and can be used to predict cluster membership. Such a combination of models is a typical feature required from an inductive database: a first query is used to mine the local models and a second query returns global models based on these local models.

Further work includes improving the computational efficiency of the approach, providing a more extensive evaluation, incorporating various constraints suitable to clustering, and integrating the approach further with inductive databases.

Acknowledgments Jan Struyf is a postdoctoral fellow of the Fund for Scientific Research of Flanders (FWO-Vlaanderen). This work was supported by the IQ project (IST-FET FP6-516169).

References

1. H. Blockeel, L. De Raedt, and J. Ramon. Top-down induction of clustering trees. In *Proc. of the 15th Int'l Conference on Machine Learning*, pages 55–63, 1998.
2. P. S. Bradley and U.M. Fayyad. Refining initial points for k–means clustering. In *Proc. of the 15th Int'l Conference on Machine Learning*, pages 91–99, 1998.
3. L. De Raedt. A perspective on inductive databases. *SIGKDD Explorations*, 4(2):69–77, 2002.
4. E. Fromont and H. Blockeel. Integrating decision tree learning into inductive databases. In *5th Int'l Workshop on Knowledge Discovery in Inductive Databases*, 2006. To appear.
5. A. Gasch, P. Spellman, C. Kao, O. Carmel-Harel, M. Eisen, G. Storz, D. Botstein, and P. Brown. Genomic expression program in the response of yeast cells to environmental changes. *Mol. Biol. Cell.*, 11:4241–4257, Dec 2000.
6. T. Imielinski and H. Mannila. A database perspective on knowledge discovery. *Communications of the ACM*, 39(11):58–64, 1996.
7. L. Kaufman and P.J. Rousseeuw, editors. *Finding groups in data: An introduction to cluster analysis.* John Wiley & Sons, 1990.

8. E. Keogh and S. Kasetty. On the need for time series data mining benchmarks: A survey and empirical demonstration. In *Proc. of the 8th ACM SIGKDD Int'l Conference on Knowledge Discovery and Data Mining*, pages 102–111, 2002.
9. S. D. Lee and L. De Raedt. An efficient algorithm for mining string data-bases under constraints. In *3th Int'l Workshop on Knowledge Discovery in Inductive Databases: Revised Selected and Invited Papers*, volume 3377 of *LNCS*, pages 108–129. Springer, 2004.
10. T. W. Liao. Clustering of time series data – a survey. *Pattern Recognition*, 38:1857–1874, 2005.
11. R. N. Mantegna. Hierarchical structure in financial markets. In *European Physical Journal*, pages 193–197, 1999.
12. R.S. Michalski and R.E. Stepp. Learning from observation: conceptual clustering. In *Machine Learning: an Artificial Intelligence Approach*, volume 1. Tioga Publishing Company, 1983.
13. J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann series in Machine Learning. Morgan Kaufmann, 1993.
14. H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spokenword recognition. In *IEEE Transaction on Acoustics, Speech, and Signal Processing*, volume ASSP-26 of *LNAI*, pages 43–49, 1978.
15. J. Sese, Y. Kurokawa, M. Monden, K. Kato, and S. Morishita. Constrained clusters of gene expression profiles with pathological features. *Bioinformatics*, 20:3137–3145, 2004.
16. I. Slavkov, S. Džeroski, J. Struyf, and S. Loskovska. Constrained clustering of gene expression profiles. In *Conference on Data Mining and Data Warehouses (SiKDD 2005) at the 7th Int'l Multi-conference on Information Society 2005*, pages 212–215, 2005.
17. J. Struyf and S. Džeroski. Constraint based induction of multi-objective regression trees. In *4th Int'l Workshop on Knowledge Discovery in Inductive Databases: Revised Selected and Invited Papers*, volume 3933 of *LNCS*, pages 222–233. Springer, 2005.
18. Ljupčo Todorovski, Bojan Cestnik, Mihael Kline, Nada Lavrač, and Sašo Džeroski. Qualitative clustering of short time-series: A case study of firms reputation data. In *ECML/PKDD'02 workshop on Integration and Collaboration Aspects of Data Mining, Decision Support and Meta-Learning*, pages 141–149. Helsinki University Printing House, August 2002.
19. B. Ženko, S. Džeroski, and J. Struyf. Learning predictive clustering rules. In *4th Int'l Workshop on Knowledge Discovery in Inductive Databases: Revised Selected and Invited Papers*, volume 3933 of *LNCS*, pages 234–250. Springer, 2005.

Integrating Decision Tree Learning into Inductive Databases

Élisa Fromont and Hendrik Blockeel

Department of Computer Science, Katholieke Universiteit Leuven,
Celestijnenlaan 200 A, 3001 Heverlee, Belgium,
{elisa.fromont, hendrik.blockeel}@cs.kuleuven.be

Abstract. In inductive databases, there is no conceptual difference between data and the models describing the data: both can be stored and queried using some query language. The approach that adheres most strictly to this philosophy is probably the one proposed by the ADReM group from Antwerp: in that approach, models are stored in relational tables and queried using standard SQL. The approach has been described in detail for association rule discovery. In this work we study how decision tree induction could be integrated in that approach. We propose a representation format for decision trees similar to the one proposed earlier for association rules, and queryable using standard SQL; and we present a prototype system in which part of the needed functionality is implemented. In the process, we identify a number of important differences between discovery of global models (such as decision trees) and local models (such as association rules), which force us to re-evaluate the motivation for the approach.

1 Introduction

An Inductive DataBase (IDB) [1] is a database that contains not only data, but also generalisations (patterns and models) valid in the data. In an IDB, ordinary queries can be used to access and manipulate data, while inductive queries can be used to generate (mine), manipulate, and apply patterns.

Two approaches have been studied to represent and query patterns and models in IDBs. First, depending on the models that will be stored, special-purpose storage and query language can be created. In this context, several researchers have proposed extensions to the popular relational query language, SQL, as a natural way to express such mining queries. For example, in [2,3], the authors have presented some extensions to SQL especially designed for mining association rules. In [4,5] the authors extend this approach to other models such as classification rules but they do not give any clues about how to actually stored those models in the IDB. In [6], De Raedt has proposed a entirely new query language based on logic and especially suited for relational data.

The second approach consists of storing the patterns and models in a straightforward way, using the usual relational database tables provided by any Relational Database Management System (RDBMS) and the standard SQL language

to represent, store and query the new generalisations made on the data. This approach is being investigated by members of the ADReM group in Antwerp[1] for frequent itemsets and association rules mining; we will refer to it in the rest of this paper as "the ADReM approach". This approach has a number of advantages over other approaches with respect to extensibility and flexibility. In this paper we investigate whether and how it can also be used for learning global models such as decision trees, and to what extent its advantages carry over to this new setting.

In Section 2 we present the basic ideas behind the ADReM approach and show how they are applied in the context of association rule discovery. In Section 3, we discuss how the same approach could be used for decision tree learning; in particular, Subsection 3.5 describes a prototype that is being implemented and shows some examples of queries that can already be used. Section 4 presents the perspectives of this work and we conclude in Section 5.

2 The ADReM Approach to Association Rule Mining

The basic idea behind the ADReM approach is that models are stored in a relational database in the same way that other information is stored: as a collection of tuples in relations or views. This idea is applicable to a broad range of models, but up till now it has been studied mostly in the context of association rules [7]. Below, we briefly review the proposed representation for association rules, the conceptual view on association rule mining that it leads to, and some implementation issues. More information on this can be found in [8].

2.1 The Conceptual View

Sets

isid	item
i1	p4
i1	p6
i1	p5
...	...
i2	red
i2	green
...	...
i3	p4
i3	1
...	...

Supports

isid	support
i0	80
i1	60
i2	40
i3	80
...	...

Rules

rid	isida	isidc	isid	conf
r1	i2	i1	i5	60%
r2	i1	i3	i6	40%
...	...	...	...	...

Fig. 1. Storing association rules

[1] {toon.calders,bart.goethals,adriana.prado}@ua.ac.be

Consider a set of transactions D. The set is often represented as a table with one row per transaction and one boolean attribute per item, but conceptually it can also be represented as a binary relational table with, for each transaction, a set of tuples of the form (*tid,item*), where *tid* is the transaction identifier and *item* is an item name. The crucial difference between the first and second representation is that in the second, the item names are values instead of attributes (hence a query can return an item name as part of its result set). Note that we are talking about the conceptual representation here — how the transaction table is really implemented is not important.

Itemsets can be represented in a similar way. Figure 1 shows the ADReM representation of frequent itemsets and association rules in an IDB. The *Sets* table represents all itemsets. A unique identifier (*isid*) is associated to each itemset (we then write that itemset as $IS(isid)$) and, for each itemset of size n, there are n rows $(isid, item_j)_{1 \leq j \leq n}$ where $item_j$ is the j^{th} item of $IS(isid)$. The *Supports* table stores the *support* of each itemset. The *Rules* table stores the association rules computed. For each rule $X \Rightarrow Y$, there is a row $(rid, isida, isidc, isid, conf)$ in the IDB where *rid* is the association rule identifier, *isida* (resp. *isidc*) is the identifier of the itemset used in the antecedent (resp. consequent) of the rule, $IS(isid) = IS(isida) \cup IS(isidc)$ and $conf$ is the confidence of the rule.

With this representation, finding association rules subject to certain constraints can be done easily using an SQL query. For instance, the query

```
select rid
from rules r
where r.conf > 0.8 and
      r.isidc in (select isid from sets where item = ''red'')
```

finds all association rules with a confidence of at least 0.8 that contain the item "red" in the consequent of the rule.

2.2 The Implementation

Conceptually, the database has tables that contain all itemsets and all association rules. But in practice, obviously, the amount of such patterns can be huge and it may be impossible to store them all. This problem is solved by keeping these tables virtual. As far as the user is concerned, those tables or *virtual mining views* contain all the tuples needed to answer the user query. In reality, each time such a table is queried, a efficient data mining algorithm is triggered by the DBMS to populate those views just sufficiently for the DBMS to be able to answer the query.

More specifically, the procedure works as follows: given a query, an execution plan is created; on the highest level this is just a relational algebra tree with tables as leaves. Standard query optimisation procedures push projection and selection conditions as deeply down into the this tree as possible, thus reducing the size of intermediate results and making the overall computation more efficient. In the context we are discussing here, the leaves may be the result of a data

mining process, and the projection/selection conditions may be pushed further down into the data mining process. Calders et al. [8] describe this optimisation process in detail.

2.3 Advantages of the Approach

The ADReM approach has several advantages over other approaches to inductive querying. The main point is that the data mining processes become much more transparent. From the user's point of view, tables with itemsets and rules etc. exist and can be queried like any other table. How these tables are filled (what data mining algorithm is run, with what parameters, etc.) is transparent. The user does not need knowledge about the many different implementations that exist and when to use what implementation, nor does she need to familiarise herself with new special-purpose query languages. The whole approach is also much more declarative: the user specifies conditions on the models that should result from the mining process, not on the mining process itself.

In the light of all these advantages, it seems useful to try a similar approach for other data mining tasks as well. In this paper, we focus on decision tree induction.

3 Integration of Decision Tree Learning

A decision tree aims at classifying instances by sorting them down the tree from the root to a leaf node that provides the classification of the instance [9]. Each node in the tree specifies a test of some attributes of the instance, and each branch descending from that node corresponds to one of the possible values for these attributes. In this paper, for simplicity reasons, we focus on decision trees with boolean attributes. Each attribute can then be seen as an item in a transaction table and transactions are instances to classify. Note that due to the well-known correspondence between trees and rule sets, a tree can be represented as a set of association rules: with each leaf corresponds one association rule, with as antecedent the conditions on the path from the root to that leaf and as consequent the majority class in that leaf. However, while the representation with one rule per leaf is interesting for prediction purposes, the structure of the tree gives more information: e.g., the higher an attribute occurs in the tree, the more important it is for the prediction. Such information is lost if we represent a tree as a set of association rules with each rule corresponding to one leaf. We will therefore choose a slightly different representation.

In this section we first discuss the motivations for integrating decision trees into IDB and we propose a representation of decision trees that will enable the user to make queries using a large number of different constraints.

3.1 Motivations

To see the motivation behind using the ADReM approach for decision tree learning, consider the current practice in decision tree induction. Given a data set,

one runs a decision tree learning algorithm, e.g., C4.5 [10], and obtains one particular decision tree. It is difficult to characterise this decision tree in any other way than by describing the algorithm. The tree is generally not the most accurate tree on the training set, nor the smallest one, nor the one most likely to generalise well according to some criterion; all one can say is that the learning algorithm tends to give relatively small trees with relatively high accuracy. The algorithm usually has a number of parameters, the meaning of which can be described quite precisely in terms of how the algorithm works, but not in terms of the results it yields. In summary, it is difficult to describe exactly what conditions the output of a decision tree fulfils without referring to the algorithm.

This situation is quite different from discovery of association rules, where the user imposes some constraints on the rules to be found (typically confidence and support) and the algorithm yields the set of all rules fulfilling these conditions. A precise mathematical description of the result set is very easy to give, whereas a similar mathematical description of the tree returned by a decision tree learner is quite impossible to give.

Are the people using decision tree learners interested in having a precise specification of the properties of the tree they find? Aren't they just interested in finding some tree with good generalisation properties and good generalisation power, without being interested in exactly how this is defined? This may be often the case, but certainly not always. Many special versions of decision tree learners have been developed: some use a cost function on attributes and try to find trees that combine high accuracy with low cost of the attributes they contain [11]; some take different misclassification costs into account when building the tree [12]; some do not aim for the highest accuracy but for balanced precision-recall [13]; etc. The fact that researchers have developed such learning algorithms shows that users sometimes do have more specific desiderata than just high predictive accuracy.

By integrating decision tree learning into inductive databases, we hope to arrive at an approach for decision tree learning that is just as precise as association rule learning: the user specifies what kind of trees she wants, and the system looks for such trees.

Here are some queries the user might be interesting in:

1. find $\{T | size(T) < 8 \wedge acc(T) > 0.8 \wedge cost(T) < 70\}$
2. find one element in $\{T | size(t) < 8 \wedge acc(t) > 0.8 \wedge cost(t) < 70\}$
3. find $\{T | size(T) < 8 \wedge (\forall T' | size(T') < 8 \Rightarrow acc(T') < acc(T))\}$
4. find $\{T | T = (t(X, t(Y, l(+), l(-)), t(Z, l(\_), l(\_)),$
   $X \in [A, B, C], Y \in [D, E], acc(T) > 0.8\}$

In the first query, the user asks for all decision trees T of *size* less than 8 nodes, of *global accuracy* higher than 0.8 and of *cost* lower than 70 (assuming that each item has a given cost). To describe the tree of interest, other criteria such as the number of misclassified examples (*error*), the *accuracy* computed for a particular class, the *precision*, the *recall*, the area under the roc curve (*auc*) for two-class problems might also be interesting.

Since the user is interested in all the trees that fulfil his criteria, the query can not be answered by triggering a standard *greedy* algorithm. Such a query implies the use of a decision tree learner that can perform an *exhaustive* search in the search space of all possible trees that fulfil these constraints. The number of such trees might be huge and this query might not be tractable. Note that without constraints, the number of decision trees that can be constructed from a database containing d attributes and a possible values for each attribute has $\prod_{i=0}^{d-1}(d-1)^{a^i}$ as a lower bound. As in the association rules case presented in section 2 we assume that the queries are constrained enough so that a realistic number of models are looked for and stored. The kind of constraints required to satisfy this criterion is still an open question.

In the second query, the user asks for one tree that fulfils some criteria. This tree can normally be computed by a regular *greedy* tree learner, though for some greedy learners there may be no guarantee that they find a valid tree if one exists.

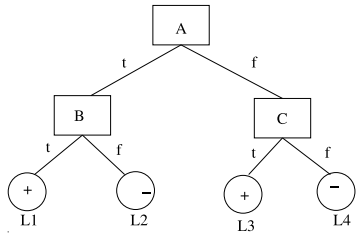
With the third query, the user is looking for the set of trees of size lower than 8 with maximal accuracy. Again, this means that the search space of trees of size lower than 8 must be exhaustively covered to ensure that the accuracy of the tree is maximal.

In the last query, the user gives *syntactic constraints* on the shape of the tree and on some attributes that must be used in the tree.

3.2 Representing Trees in a Relational Database

The virtual mining view that holds the predictive models should be precise enough to enable the user to ask SQL queries as easily as possible without having to design new keywords for the SQL language. Figure 2 shows the database framework we propose for integrating decision trees into IDB. We use the table presented in section 2 to represent the data. We assume in the following that all the data-dependent measures (such as accuracy) are referring to these data. The decision trees generated from the data D can be stored in the same database as D and as the possible association rules computed from D, by using the following schema:

1. The *Tree_sets* table is inspired by the *Sets* table created for association rules. We choose to represent a node of a tree by the itemset that characterises the examples that belong to this node. For instance, if the itemset is $A\overline{B}$, examples in this node must fulfil the criteria $A = true$ and $B = false$. In association rules, only the presence of certain items is required: there is no condition that specifies the absence of an item. To cope with association rules derived from trees such that the one corresponding to the leaf $L2$ of the tree given in the figure 2 ($A\overline{B} \Rightarrow -$), a *sign* attribute is thus added to the table to indicate wether the presence (1) or the absence (0) of an item is required.
   As in the *Sets* table, a unique identifier ($isid$) is associated to each itemset and, for each itemset of size n, there are n rows $(isid, item_j, sign_j)_{1 \leq j \leq n}$



Trees_charac

treeID	size	error	accuracy	auc	cost	...
T1	...	...	...	...	...	...
T2	...	...	...	...	...	...
...	...	...	...	...	...	...

Tree_sets

isid	item	sign
i0	$\emptyset$	1
i1	A	1
i2	A	1
i2	B	1
l1	A	1
l1	B	1
l1	+	1
i3	A	1
i3	B	0
l2	A	1
l2	−	1
l2	*B*	0
l3	...	...
l4	...	...
...	...	...

all_trees

treeID	isid	leaf
T1	i1	0
T1	i2	0
T1	L1	1
...	...	...
T1	L2	1
...	...	...
T1	L3	1
...	...	...
T1	L4	1
T2	...	...
T3	...	...
...	...	...

greedy_trees

treeID	isid	leaf
T1	i1	0
T1	i2	0
T1	L1	1
...	...	...
T1	L2	1
...	...	...
T1	L3	1
...	...	...
T1	L4	1

Fig. 2. Storing decision trees

where $item_j$ is the j^{th} item of the itemset identified by $isid$ and $sign_j$ is its sign. $i0$ stands for the empty itemset.

2. The *all_trees* and *greedy_trees* tables give a precise description of each tree in terms of the itemsets from *Tree_sets*. A unique identifier ($treeID$) is associated to each tree and each itemset corresponding to a node in this tree is associated to his $treeID$. A boolean attribute $leaf$ differentiates internal nodes of the tree ($leaf = 0$) from the leaves ($leaf = 1$). At each level k of the tree, nodes are composed by k-itemsets. The *all_trees* table is supposed to hold all possible trees, whereas the *greedy_trees* table holds an implementation-dependent subset of all trees (more specifically those trees that might be found by greedy learners under certain conditions and constraints).
3. The *Trees_charac* table gives all the characteristics of the tree the user might be interested in. For each tree identified by $treeID$ corresponds a row $(size, error, accuracy, cost, auc)$ that are the computed characteristics of the tree (see section 3.1).

3.3 Querying Decision Trees Using Virtual Views

The structure created is sufficient to make some interesting queries on the data, provided that the data mining algorithms connected to the database compute the different characteristics of the association rules or of the trees that hold in the IDB. The following queries are examples of what can be done on such database :

```
SELECT trees_charac.* FROM trees_charac, all_trees
WHERE trees_charac.treeID = all_trees.treeID AND
 accuracy > 0.8 and size < 8;
```

This query selects the characteristics of all trees that can be computed from the database with accuracy higher than 0.80 and size lower than 8.

```
SELECT treeID FROM trees_charac, greedy_trees
WHERE trees_charac.treeID = greedy_trees.treeID
and trees_charac.error < 10;
```

This query selects a greedy-computed tree with an error lower than 10.

```
SELECT trees_charac.* FROM trees_charac, all_trees
WHERE trees_charac.treeID = all_trees.treeID
AND accuracy = (select max(accuracy) from trees_charac);
```

This query selects the characteristics of the tree with maximum accuracy from all the possible computed trees.

```
SELECT treeID FROM greedy_trees, tree_sets
WHERE greedy_trees.isid = tree_sets.isid
AND (tree_sets.isid
     IN (select isid from tree_sets where item = ''A''));
```

This query selects a greedy-computed tree which contains the item "A".

3.4 Querying Decision Trees Using Itemsets

The framework is flexible enough to allow queries with constraints on metrics that were not included in the virtual view from the beginning. The user can create his own virtual mining view using information on the support of the itemsets. We illustrated this with the notions of accuracy and size.

The accuracy of a specific leaf in the tree can be computed from the support of the itemsets that belong to the leaf [14] using the formula (on figure 2):

$$acc(L_1) = \frac{support(+AB)}{support(AB)} \ldots acc(L_2) = \frac{support(-A\overline{B})}{support(A\overline{B})}.$$

The mean accuracy of each leaf is the global accuracy of the tree. This can be computed without any information on the actual structure of the tree using the

formula (on figure 2):

$$acc(T) = \frac{acc(L1) * support(AB) + acc(L2) * support(A\overline{B}) + \ldots}{support(\emptyset)}$$
$$= \frac{support(+AB) + support(-A\overline{B}) + \ldots}{support(\emptyset)}.$$

Some itemsets do not have any support associated with because they include "negative" item. In this case, some formula such as:

$$support(A\overline{B}-) = support(A-) - support(AB-)$$

can be used to compute the support of all itemsets from the support of the "positive" itemsets [15].

These formulas can be translated into the SQL language to compute all the characteristics in the *tree_charac* table. We consider that, as in Section 2, we have a Supports table that countains the support of all itemsets.

```
acc(T1)= SELECT SUM(Supports.support) /
(SELECT Supports.support FROM Supports
WHERE Supports.isid = ''I0'') as accuracy
FROM Supports, all_trees
WHERE Supports.isid = all_trees.isid
AND all_trees.treeID = T1
GROUP BY all_trees.treeID

size(T1) = SELECT COUNT(*) FROM all_trees
WHERE all_trees.treeID = T1
```

3.5 Implementation

An Apriori-like algorithm for association rule mining was connected to a standard Oracle Database by the ADReM group to use constraints such as the support of the itemsets, the confidence of the rules and the presence or absence of some item in the resulting association rules. We connected to the same system a decision tree learner named *Clus*. Clus is a predictive clustering tree learner developed by Jan Struyf that uses a standard recursive top-down induction algorithm to construct decision trees. First, a large tree is built based on the training data then the tree is pruned according to some user constraints. The constraints described by [16] were implemented in this generic and efficient system [17] so it currently supports constraints on the size of the tree (number of nodes), on the error of the tree and on the syntax of the tree. The error measure used for classification trees learning is the number of incorrectly predicted classes. The syntactic constraints allow the user to introduce expert knowledge in the tree by specifying a partial tree, i.e, a subtree including the root and so the most important attributes of the tree. Other constraints discussed in section 3.1 have to be implemented in the system. For the moment, queries such as the following one can be used:

```
SQL> select * from trees_charac where err < 8 and sz= 9;

TREE_ID SZ ERROR ACCURACY
------- -- ----- --------
      0  9     3     0,98
1 rows selected.


SQL>select * from trees_charac where err < 8 and sz <= 8;

TREE_ID SZ ERROR ACCURACY
------- -- ----- --------
      1  7     4    0,973
1 rows selected.

SQL> select * from trees_charac where sz< 4;

TREE_ID SZ ERROR ACCURACY
------- -- ----- --------
      2  3    50    0,667
1 rows selected.
```

All the trees computed with the different queries could be stored in a "log" table that can be queried just as easily. After the session above, this table would contain:

TREE_ID	SZ	ERROR	ACCURACY
0	9	3	0,98
1	7	4	0,973
2	3	50	0,667

4 Perspectives

There are many open problems related to the proposed approach. For instance, for efficiency reasons, the system should be able to look at the "log" table that contains the previously computed trees to check if the answer of the current query has not already been computed before triggering a data mining algorithm. If the user asks for all trees of size less than 8 and then later for all trees of size less than 6, the results computed from the first query should be reusable for the second query. The "log" table should then also contain the previously asked queries together with the computed trees, which raises the question of how to store the queries themselves in the database. This entire problem, called *interactive mining* because it refers to the reutilisation of queries posed within the same working session, has been investigated for association rules [18], but not yet for decision tree learning.

Another type of problem occurs if the database has been modified between two queries. Is it possible to use some previously computed predicted models to

compute more efficiently new predictive models from a modified database? This problem known has *incremental learning* has already been studied for decision trees [19] when a new example is added to the database.

These functionalities has to be integrated into the prototype along with the extension of the framework to multi-valued attributes.

Besides, as predictive models ultimately aim at predicting the class of new examples, it would be interesting to include that possibility in the IDB. This is currently non-trivial in our approach, it requires complicated queries.

Generally, the limitations of our approach with respect to what can be expressed, and how difficult it is to express it, are still unclear. With respect to query complexity, it may be useful to consider an extended relational model where trees are an abstract data type with a number of predefined operators, instead of being stored as sets of tuples.

Another perspective will be the integration of other predictive models such as *Bayesian Network* in the same IDB framework already designed for association rules and decision trees mining. The user might be interested in query such as "find the Bayesian network of size 3 with maximal probability". Again, a structure to store Bayesian networks has to be designed and algorithm than can build Bayesian networks under constraints has to be implemented.

5 Conclusion

In this paper we have studied how decision tree induction could be integrated in inductive databases following the ADReM approach. Considering only boolean attributes, the representation of trees in a relational database is quite similar to that of association rules, with this difference that the conjunctions describing nodes may have negated literals whereas itemsets only contain positive literals. A more important difference is that a decision tree learner typically returns one tree that is "optimal" in some not-very-precisely-defined way, whereas the IDB approach lends itself more easily to mining approaches that return all results fulfilling certain well-defined conditions. It is therefore useful to introduce a *greedy_trees* table in addition to the *all_trees* table, where queries to *greedy_trees* trigger execution of a standard tree learner and queries to *all_trees* trigger execution of an exhaustive tree learner. We have described a number of example queries that could be used, presented a preliminary implementation that handles such queries, and discussed open questions and perspectives of this work.

Acknowledgement

Hendrik Blockeel is a post-doctoral fellow of the Fund For Scientific Research of Flanders (FWO-Vlaanderen). This work is funded through the GOA project 2003/8, "Inductive Knowledge bases", and the FWO project "Foundations for inductive databases". The authors thank Jan Struyf, Sašo Džeroski and the ADReM group for many interesting discussions, and in particular Jan for his help with the Clus system and Adriana Prado for her help with the IDB prototype.

References

1. Imielinski, T., Mannila, H.: A database perspective on knowledge discovery. Comm. Of The Acm **39** (1996) 58–64
2. Meo, R., Psaila, G., Ceri, S.: An extension to sql for mining association rules. Data Min. Knowl. Discov. **2** (1998) 195–224
3. Imielinski, T., Virmani, A.: Msql: A query language for database mining. Data Min. Knowl. Discov. **3** (1999) 373–408
4. Kramer, S., Aufschild, V., Hapfelmeier, A., Jarasch, A., Kessler, K., Reckow, S., Wicker, J., Richter, L.: Inductive databases in the relational model: The data as the bridge. In: KDID. (2005) 124–138
5. Han, J., Fu, Y., Wang, W., Koperski, K., Zaiane, O.: DMQL: A data mining query language for relational databases. In: SIGMOD'96 Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD'96), Montreal, Canada (1996)
6. De Raedt, L.: A logical database mining query language. In Cussens, J., Frisch, A., eds.: ILP00. Volume 1866 of LNAI., SV (2000) 78–92
7. 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
8. Calders, T., Goethals, B., Prado, A.: Integrating pattern mining in relational databases. In: PKDD: 10th European Conference on Principles and Practice of Knowledge Discovery in Databases. LNCS, Springer (2006) To appear
9. Mitchell, T.: Machine Learning. McGraw-Hill, New York (1997)
10. Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann (1993)
11. Turney, P.: Cost-sensitive classification: Empirical evaluation of a hybrid genetic decision tree induction algorithm. Journal of Artificial Intelligence Research **2** (1995) 369–409
12. Domingos, P.: Metacost: A general method for making classifiers cost-sensitive. In: Knowledge Discovery and Data Mining. (1999) 155–164
13. Xiaobing, W.: Knowledge representation and inductive learning with xml. In: WI '04: Proceedings of the Web Intelligence, IEEE/WIC/ACM International Conference on (WI'04), Washington, DC, USA, IEEE Computer Society (2004) 491–494
14. Pance, P., Dzeroski, S., Blockeel, H., Loskovska, S.: Predictive data mining using itemset frequencies. In: Zbornik 8. mednarodne multikonference Informacijska druzba. Ljubljana: Institut "Jozef Stefan", Informacijska druzba (2005) 224–227
15. Calders, T., Goethals, B.: Mining all non-derivable frequent itemsets. In: Proceedings of the 6th European Conference on Principles of Data Mining and Knowledge Discovery. Volume 2431 of LNCS., Springer-Verlag (2002) 74–85
16. Garofalakis, M.N., Hyun, D., Rastogi, R., Shim, K.: Building decision trees with constraints. Data Min. Knowl. Discov. **7** (2003) 187–214
17. Struyf, J., Dzeroski, S.: Constraint based induction of multi-objective regression trees. In: KDID. (2005) 222–233
18. Goethals, B., den Bussche, J.V.: On supporting interactive association rule mining. In: Proceedings of the Second International Conference on Data Warehousing and Knowledge Discovery. Volume 1874 of LNCS., Springer (2000) 307–316
19. Utgoff, P.E.: Incremental induction of decision trees. Machine Learning **4** (1989) 161–186

An Integrated Multi-task Inductive Database and Decision Support System VINLEN: An Initial Implementation and First Results

Kenneth A. Kaufman, Ryszard S. Michalski, Jaroslaw Pietrzykowski, and Janusz Wojtusiak

Machine Learning and Inference Laboratory, George Mason University
{kaufman, michalski, jarek, jwojt}@mli.gmu.edu

Abstract. A brief review of the current research on VINLEN multitask inductive database and decision support system is presented. VINLEN integrates a wide range of knowledge generation operators that given input data and/or knowledge create new knowledge. The central operator of VINLEN is a natural induction module that generates hypotheses from data in the form of attributional rules. Such rules are easy to understand and interpret because they directly correspond to equivalent natural language descriptions. This operator is illustrated by an application to discovering relationships between lifestyles and diseases in men. The conclusion outlines plans for future research.

1 Introduction

This paper briefly reviews our current research on the development of VINLEN, a multitask inductive database and decision support system. In VINLEN, inductive inference capabilities are integrated with a database and a knowledge base. Standard relational database operators, implemented through an SQL client, are combined with *knowledge generation operators* (KGOs) using *Knowledge Query Language* (KQL).

KGOs operate on knowledge segments, consisting of a combination of one or more relational tables linked to relevant knowledge component in the VINLEN knowledge base. A KGO takes as input knowledge segments, and generates an output knowledge segment.

Two important capabilities are required from knowledge generation operators: (1) that their results are in a form easy to understand and interpret by users, (2) that KGO-generated results are in forms that can be accepted as input by *compatible* KGO operators. A compatible operator is the one that can use that knowledge if it is submitted to it.

The central knowledge generation operator in VINLEN is implemented in the natural induction module that creates inductive generalizations of or discovers patterns in data in forms that appear natural to people, by employing *attributional calculus* as a representation language [8]. Attributional calculus is a logic system that combines elements of propositional, predicate, and multiple-valued logics for facilitating induc-

tive inference. It serves both as an inference system and as knowledge representation formalism.

Attributional descriptions, the primary form of knowledge representation in VINLEN, are more expressive than conventional decision rules that use only <attribute-relation-value> conditions. Attributional rules use conditions that may involve more than one attribute and relate them to a subset of values or to other attributes. Section 3 gives more details on this topic.

2 An Overview of VINLEN

Research on the VINLEN system grows out of our previous efforts on the development of INLEN, an early system deeply integrating databases and inductive learning capabilities for the purpose of multistrategy learning, data mining, and decision support e.g. [9]. The advantages such integration and of inductive databases are now being widely recognized, as evidenced by this workshop and earlier efforts e.g. [2, 4, 5].

VINLEN represents a step beyond the approach to inductive databases taken by some authors, namely, it not only integrates database and a knowledge base containing selected results of inductive inference (using the capabilities of database), but also is a host of inductive and deductive inference operators, as well as various other data analysis and pattern discovery capabilities. It supports inferences resulting from a series of applications of its operators according to a script in *knowledge query language* (KQL). This way it can automatically conduct experiments that involve creating, storing and managing of the relevant data and knowledge laying the groundwork for higher levels of sophistication in inductive databases functionality based, for example, on a meta-learning approach. Therefore, it provides a powerful tool for conducting experiments and may avoid some pitfalls resulting from too limited exploration of data or from the parameters of the methods [1].

An important concept in VINLEN is that of a *knowledge system* that consists of a database, which can be local or distributed, and a knowledge base. The term "knowledge system" signifies a system integrating a database and a relevant knowledge base to support knowledge mining and knowledge application to the problem at hand. A knowledge base contains handcrafted knowledge and results of applying a range of knowledge generation and management operators to data in the database and/or to prior knowledge in the knowledge base.

The prior knowledge contains definitions of the domains and types of attributes in the database, data constraints, value hierarchies of structured attributes, known relationships binding attributes, and any other background knowledge that users may have represented in the system. During the operation of an inductive database, the knowledge base is populated by newly generated data descriptions, hypothetical patterns, data classifications, statistical information, results from hypothesis testing, etc.

The data in each VINLEN knowledge system are stored internally in relational tables, as are other system components. Most of the entities utilized by the system, such as events (a.k.a. examples), datasets, attributes, attribute domains, rule conditions, rules, rulesets, and classifiers, are presented in individual tables in the database, and connected via relations.

Events are stored in an event table, which is populated either from external source, manually by the user, or by a VINLEN operator, for example, by an operator that selects most representative events from the training dataset. In addition to regular attribute values, events may contain meta-values, such as "unknown", "irrelevant" and "not-applicable" which require a special handling during the learning or knowledge application processes [10]. The "unknown" values, denoted by a "?", represent cases when a regular attribute value exists, but is unknown for some reason, the "irrelevant" and "non-applicable" values represent domain knowledge provided by the user.

Each event may carry additional meta-information, such as *event significanc*e and *event frequency*. The event significance is a value assigned to an event by the user or by the program to represent some form of importance of the event for problem at hand. For different types of problems it may have a different meaning. For example, in concept learning, it may represent the typicality of the event; in optimization via evolutionary computation, it represents the value of the fitness function for that event. Event frequency is a number of occurrences of the given event in the training or testing data.

Prior knowledge and knowledge generated by VINLEN are stored in a hierarchy of relational tables that can be used by KGOs. Rule-based classifiers learned from the data are in the form of families of attributional rulesets. Their components, such as selectors (conditions), complexes (conjunctions of conditions), exceptions, single rules, rulesets, and alternative rulesets are considered as individual entities, and as such are represented by separate tables connected by relations. Parameter sets for individual operators are stored in method-specific tables. This storage methodology facilitates an efficient access to all components of the classifiers through a standard SQL interface.

A *Target Knowledge Specification* is a generalization of a database query; specifically, it is a user's request for a knowledge segment to be created by the system, based on the data and knowledge already present. The core of VINLEN consists of *Inductive Database Operators*, which call upon various programs for knowledge generation (e.g., rule learning, conceptual clustering, target data generation, optimization, etc.), as well as data management. These operators can be invoked by a user directly, or through KQL.

To provide a general overview and easy access to all VINLEN operators, we have developed a visual interface that consists of VINLEN views at different abstraction levels. Fig. 1 presents the most abstract view of the main panel of VINLEN.

The central part contains icons for managing database (DB), knowledge base (KB), and knowledge systems (KS). By clicking on DB, KB, or KS, the user can select and access database, knowledge base, and knowledge system that are available to VINLEN. Each of the rectangular buttons allows the user to access a family of knowledge generation operators of a given type.

For example, the button "Learn Rules" allows one to access operators that learn ordinary attributional rules, rules with exceptions, multi-head rules, and rule-trees. A similar multi-function role have other buttons, such as "Access attributes", "Improve rules", "Learn trees", "Create clusters", "Access scout", "Define dataset", etc.

All operators are integrated through Knowledge Query Language (KQL) that is an extension of the SQL database query language. In addition to conventional data management operators, KQL includes operators for conducting inductive and deductive

inference, statistical analysis, application and visualization of learned knowledge, and various supportive functions. KQL allows a user to define *knowledge scouts* that are KQL scripts for automatically executing a series of knowledge generation operators in search for knowledge of interest to the user.

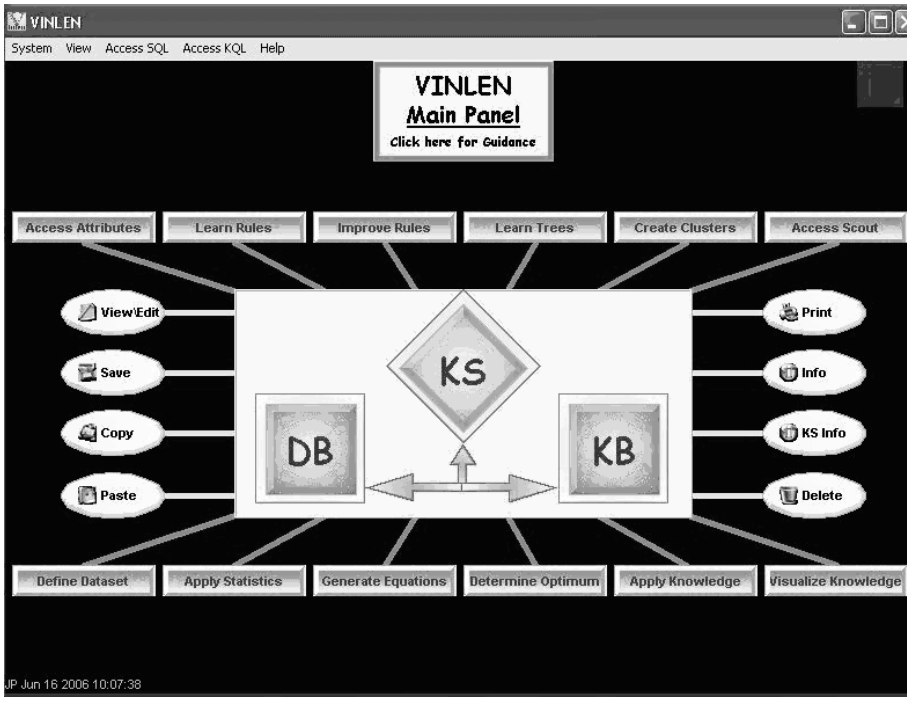


Fig. 1. The front panel of VINLEN (a black and white version of the original)

Due to a wide range of capabilities and the types of operators it involves, many of which are unique to VINLEN, KQL is very different from other high-level languages developed for data exploration, many of which have been Prolog-based.

Among the non-Prolog-based languages, M-SQL [6] is philosophically somewhat similar to KQL in that it builds upon the SQL data query language. It integrates in it, only one inductive operator, in contrast to VINLEN that adds to it a wide range of operators. In [7], the inductive the M-SOL operator is used to analyze users' internet activity. KQML [3] allows the user to query for specific pieces of knowledge, but it does not support multiple discovery operators and the abstract templates that are available in KQL.

A somewhat related to KQL is also a query language presented in [2] that has the capability for specifying the type of knowledge to search for, e.g., rules with the confidence level above a given threshold. Being a Prolog-based language, it has the capability for directly expressing relational descriptions, but does not involve such a wide range and versatile operators that VINLEN does. While VINLEN can also search for rules with a confidence above a certain threshold, it can also seek rules with maximum confidence.

3 VINLEN Operators

As mentioned earlier, VINLEN aims at providing user with an extensive set of different knowledge generation and data management operators, and with a language, KQL, to develop scouts for executing sequences of these operators. Such scouts can thus automatically perform knowledge discovery experiments.

Basic functionality of VINLEN allows the user to browse, edit, copy, delete, print, define, import and export data and knowledge. More advanced functions support data selection, attribute evaluation and selection, attribute discretization, and estimation of parameter settings for operators to achieve a desired result, ,and a range of learning and knowledge discovery functions. The rule learning module is based on the AQ21 program for natural induction [15].

As mentioned earlier, VINLEN's knowledge representation is based on attributional calculus [8]. The basic unit of knowledge representation is an attributional rule in the form:

Consequent <= *Premise* |_ *Exception*,

where *Consequent*, *Premise*, and *Exception* are *conjunctive descriptions*, or *complexes*, which are conjunctions of *attributional conditions*. An attributional condition (a.k.a. *selector*) can be viewed as equivalent to a simple natural language statement. Its general form is:

[*L relsym R*],

where:

L (the left side or referent) contains one attribute, or several attributes joined by "&" or "v", called internal conjunction and disjunction, respectively. L can also be one of the standard *derived attributes*: count, max, min, and avg.

R (the right side or reference) is an expression specifying a value or a subset of values from the domain of the attribute(s) in L. If the subset contains values of a nominal (unordered) attribute, they are joined by the symbol "v" (called internal disjunction); if the subset contains consecutive values of a linear attribute, they are represented by joining the extreme values by operator "..", called range. R can also be a single attribute of the same type as the attribute or attributes in L.

relsym is a relational symbol from the set: {=, ≠, >, ≥, <, ≤}. Relational operators {=, ≠} apply to all types of attributes. Relations {>, ≥, <, ≤} apply only to linear attributes.

Brackets [ ], may be omitted, if their omission causes no confusion. If brackets are used, the conjunction of two selectors is usually written as their concatenation. If an attribute, x, is binary, the condition [x = 1] can be written simply as the literal x, and [x = 0] as the literal ~x. Thus, if attributes are binary, attributional conditions reduce to propositional literals.

An attributional condition is called *basic*, if its left side, L, is a single attribute, the relational symbol is one of { =, >, ≥, <, ≤}, and the right side, R, is a single value; otherwise, it is called *extended*.

Examples of basic conditions:

[x1 = 1], alternatively, x1 (The value x1 is 1)
[x1 = 0], alternatively, ~ x1 (The value x1 is 0, the alternative notations assume that x1 is binary)
[color = red] (The color is red)
[length < 5"] (The length is smaller than 5 inches)
[temperature ≥ 32° C] (The temperature is greater than or equal to 32° C)
[tools={mallet, knife}] (The tools are mallet and knife)

Examples of extended conditions:

[color = red v blue v green] (The color is red, blue or green)
[blood-type ≠ A] (The blood type is not A)
[length= 4..12] (The length is between 4 and 12, inclusive)
[color ≠ green] (The color is not green)
[height > width] (The height is greater than the width)
[height v width < 3 m] (The height or the width is smaller than 3 m)
[height & width ≥ 7 cm] (The height and width are both at least 7 cm)
[height & width < length] (Both the height and the width are smaller than the length)

Operators "v" and "&" when applied to non-binary attributes or to their values are called internal disjunction and conjunction, respectively.

A set of attributional rules with the same consequent (e.g., indicating the same class) is called an *attributional ruleset*. A set of attributional rulesets whose consequent spans all values of an output (dependent variable) is called an *attributional classifier*.

The design of VINLEN includes the following learning and inference capabilities:

- Learning complete and consistent attributional classifiers
- Optimizing attributional classifiers
- Discovering strong patterns in data (attributional rules that represent strong regularities but may be partially inconsistent with the data)
- Generation of multi-head attributional rules (with more than one attribute in the consequent of a rule);
- Deriving optimized decision trees from attributional classifiers e.g. [13];
- Applying attributional classifiers to data, and evaluating the results in the case of testing data;
- Discovering conceptual clusters in data e.g. [11];
- Determining the optimum of a given function using non-Darwinian evolutionary computation [14];

To facilitate the interpretability and understandability of learned knowledge, VINLEN includes operators that visualize knowledge in the form of *concept association graphs* and *generalized logic diagrams* e.g. [12].

It should be mentioned that although all the knowledge generation operators described above have been developed, implemented, and tested individually, so far only some of them have been fully integrated in VINLEN. The process of integration of so many operators, and developing an appropriate graphical user interface for each them is a very labor-intensive effort, and it will take some time before it is completed.

4 Knowledge scouts

As mentioned earlier, VINLEN operators (learning and inference operators, as well as data and knowledge management operators) can be arranged into knowledge scouts. A knowledge scout is a KQL script that for automatically applying various operators in search of *target knowledge* in the database. The target knowledge is defined abstractly by specifying properties of pieces of knowledge that are of interest to the given user (or a specified group of users). Simple examples of target knowledge specification are "Determine a general classifier from the dataset DS, that maximizes the criterion of attributional classifier ACQ", or "Determine a conceptual clustering of the dataset DS that optimizes the clustering quality criterion CCQ."

Here is very simple example of a knowledge scout in the form of a one-line KQL script:

```
CREATE RULES GDP-Classifier TYPE CC FROM TR1 USING AQ21
WHERE decision is "GDP", searchscope = 3
```

This script instructs VINLEN to apply a rule learning operator to the data set TR1, using the AQ21 module with the *searchscope* parameter set to 3 (the width of the beam search used in the learning module). Since settings of other parameters are not specified, the default values will be used. The goal of applying the learning operator is to learn a consistent and complete (CC) classifier for the output attribute "GDP" The classifier is to be stored in the knowledge base under the name "GDP-Classifier."

In order to synthesize target knowledge, a knowledge scout may consist of many lines of KQL code that request an execution of a sequence of KQL operators involving data, intermediate results, previously learned knowledge and background knowledge. The latter may include the types of attributes, their domains (including hierarchies of structured attributes), problem constraints, and rules for constructing derived attributes. At every step of running knowledge scout, an application of one operator may depend on the results of previous operators. This is possible due to the inclusion of tests of properties of data and knowledge components, of the results of their application to data, and the use of a branching operator in KQL. For example, a condition for repeating a learning operator may be:

"If the average consistency of attributional rules in the classifier is smaller that .95 or the number of rules in the classifier is greater than 10, the accuracy of the classifier

on the testing data is smaller than .93, and the number of learning runs is smaller than 50, repeat the run with the *searchscope* 15, otherwise, return the results."

5 An Example of Application to a Medical Domain

This section illustrates an application of the learning module of VINLEN to a problem of determining relationships between lifestyles and diseases of non-smoking males, aged 50-65, and displaying results in the form of a concept association graph. The study employed a database from the American Cancer Society that contained 73,553 records of responses of patients to questions regarding their lifestyles and diseases. Each patient was described in terms of 32 attributes: 7 lifestyle attributes (2 Boolean, 2 numeric, and 3 rank), and 25 Boolean attributes denoting diseases. The learning operator determined patterns (approximate attributional rules) characterizing the relationships between 25 diseases and the lifestyles and other diseases. Fig. 2 shows one example of discovered patterns (*HBP* stands for High Blood Pressure, *Rotundity* is a discretized ratio of the patient's weight to his height and *YinN* denotes the years the patient lived in the same neighborhood).

[Arthritis = Present] <=
    [HBP=present: 432, 1765] &
    [Education<=college_grad: 940, 4529] &
    [Rotundity>=low: 1070, 5578] &
    [YinN > 0: 1109, 5910]: p = 325, n = 1156; P = 1171, N = 6240

Fig. 2. A pattern for Arthritis discovered in the medical database.

The two numbers listed within each condition after the colon denote the numbers of positive and negative examples in the training set covered by that condition, respectively; p and n, are the numbers of positive and negative examples in the training set covered by the entire rule, respectively; and P and N are the numbers of positive and negative examples in the training data for that class (here, Arthritis), respectively.

The pattern in Fig. 2 defines a set of conditions under which patients had arthritis relatively frequently. These conditions include the presence of high blood pressure, no education beyond college, more than "very low" rotundity, and the patient's having lived in his current neighborhood at least one year. In the training data, about 16% of the patients had arthritis (P / (P + N)), but among patients satisfying the pattern, the percentage grows to 22% (p / (p + n)). The most significant factor in the pattern is high blood pressure, which by itself has consistency of about 19%. The discovered patterns were visualized using a concept association graph. Fig. 3. presents one such graph that was automatically generated using the CAG visualization operator.

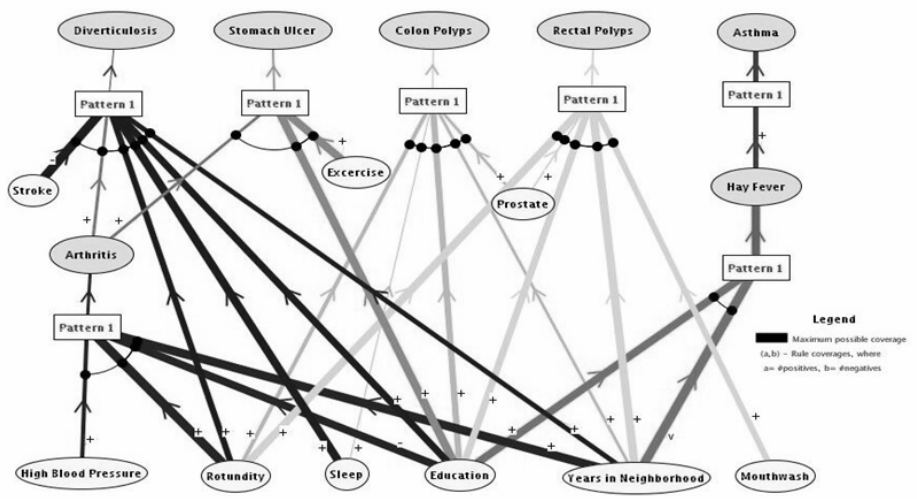


Fig. 3. Concept Association Graph representing seven patterns in the medical database.

The thickness of links in CAG reflects the condition's consistency, and its annotation (+, –, v, or ^) indicates the type of the relationship between condition and consequent. Specifically, "+" and "-" represent a positive and negative monotonic relationship, respectively; and "v" and "^" indicate that extreme values of the attribute indicate higher or lower values of the consequent attribute, respectively.

While no claim is made as to the medical validity and significance of these results, they indicate, however, that the developed methodology is potentially capable of discovering important patterns in the data and representing them in an understandable way, either as qualitative relationships in the form of association rules, or graphically via a concept association graph.

6 Summary and Future Work

The paper described current research on VINLEN, a large-scale system for integrating operators for data management, data analysis, knowledge discovery for classification, clustering and optimization, for data and knowledge visualization, knowledge testing and application, and decision support. The underlying knowledge representation is based on attributional calculus that combines features of propositional, predicate, and multi-valued logic for the purpose of facilitating knowledge discovery.

Individual operators can be invoked by the user individually, via graphical user interface, or automatically, via a knowledge scout, which is a script in high level language, called knowledge query language (KQL). KQL is an extension of SQL that adds to it various knowledge generation, management, visualization and application operators.

The system is still under development. In this paper, we focused on two central operators already implemented in VINLEN, namely, the operator for learning attributional classifiers, and the operator for visualizing such classifiers using concept asso-

ciation graphs. These operators have been illustrated by an example in medical domain.

Other operators, such conceptual clustering, intelligent target data and generation, function optimization via Learnable Evolution, and database manipulation through an SQL client have been partially implemented. Various statistical operators, modules for applying knowledge to data for generating decisions, and a mechanism for creating knowledge query language scripts to guide data exploration tasks are under development.

The main contribution of the VINLEN project is the development of a general methodology for a tight integration of a database, knowledge base, data management operators, knowledge generation operators, a knowledge query language, and a user-oriented visual interface. Research on VINLEN aims at achieving methodological advances and for developing new tools for knowledge mining in databases, and for the decision support based on the knowledge discovered.

Acknowledgments

Research described here has been conducted in the Machine Learning and Inference Laboratory at George Mason University and has been supported in part by the National Science Foundation under Grants No. IIS-9906858 and IIS-0097476, and in part by the UMBC/LUCITE #32 grant. In a few cases, presented results have been obtained under earlier funding from the National Science Foundation, the Office of Naval Research, or the Defense Advanced Research Projects Agency. The findings and opinions expressed here are those of the authors, and do not necessarily reflect those of the above sponsoring organizations.

References

1. Blockeel, H.: Experiment Databases: A Novel Methodology for Experimental Research. In: Bonchi, F., Boulicaut, J-F. (eds.): Knowledge Discovery in Inductive Databases. 4th International Workshop, KDID'05, Revised, Selected and Invited Papers. Lecture Notes in Computer Science, Vol. 3933. Springer-Verlag, Berlin Heidelberg New York (2006) 72-85
2. De Raedt, L.: A Perspective on Inductive Databases. ACM SIGKDD Explorations Newsletter, Vol. 4, Issue 2. ACM Press, New York, NY, USA (2002) 69-77
3. Finin, T., Fritzson, R., McKay, D. and McEntire, R.: KQML as an Agent Communication Language. Proceedings of the Third International Conference on Information and Knowledge Management, CIKM'94. ACM Press (1994) 456-463
4. Flach, P. and Dzeroski, S.: Editorial: Inductive Logic Programming is Coming of Age. Machine Learning, Vol. 44, Number 3. Springer Netherlands (2001) 207-209
5. Hätönen, K. Mika Klemettinen, M., Miettinen, M.: Remarks on the Industrial Application of Inductive Database Technologies. In: Boulicaut, J-F., De Raedt, L., Mannila, H. (eds.): Constraint-Based Mining and Inductive Databases, European Workshop on Inductive Databases and Constraint Based Mining, Hinterzarten, Germany, March 11-13, 2004, Revised Selected Papers. Lecture Notes in Computer Science, Vol. 3848. Springer (2005) 196-215
6. Imielinski, T., Virmani, A. and Abdulghani, A.: DataMine: Application Programming Interface and Query Language for Database Mining. In: Proceedings of the 2nd International

Conference on Knowledge Discovery and Data Mining, KDD'96. AAAI Press (1996) 256-261
7. Meo, R., Vernier,F., Barreri, R., Matera, M. and Carregio, D.: Applying a Data Mining Query Language to the Discovery of Interesting Patterns in WEB Logs. Workshop on Inductive Databases and Constraint Based Mining, Hinterzarten, Germany (2004)
8. Michalski, R.S.: ATTRIBUTIONAL CALCULUS: A Logic and Representation Language for Natural Induction. Reports of the Machine Learning and Inference Laboratory, MLI 04-2, George Mason University, Fairfax,, VA (April 2004)
9. Michalski, R.S., Kerschberg, L., Kaufman, K. Ribeiro, J.: Mining For Knowledge in Databases: The INLEN Architecture, Initial Implementation and First Results. Intelligent Information Systems: Integrating Artificial Intelligence and Database Technologies, Vol. 1, No. 1 (August 1992) 85-113
10. Michalski, R.S., Wojtusiak, J.: Reasoning with Meta-values in AQ Learning. Reports of the Machine Learning and Inference Laboratory, George Mason University, Fairfax, VA (June 2006)
11. Seeman, W.D., Michalski, R. S.: The CLUSTER3 System for Goal-oriented Conceptual Clustering: Method and Preliminary Results. Proceedings of The Data Mining and Information Engineering Conference, Prague, Czech Republic, July 11-13, 2006.
12. Sniezynski, B., Szymacha, R., Michalski, R. S.: Knowledge Visualization Using Optimized General Logic Diagrams. Proceedings of the Intelligent Information Processing and Web Mining Conference, IIPWM 05, Gdansk, Poland, June 13-16, 2005.
13. Szydlo, T., Sniezynski, B., Michalski, R. S.: A Rules-to-Trees Conversion in the Inductive Database System VINLEN. Proceedings of the Intelligent Information Processing and Web Mining Conference, IIPWM 05, Gdansk, Poland, June 13-16, 2005.
14. Wojtusiak, J., Michalski, R. S.: The LEM3 Implementation of Learnable Evolution Model and Its Testing on Complex Function Optimization Problems. Proceedings of Genetic and Evolutionary Computation Conference, GECCO 2006, Seattle, WA, July 8-12, 2006.
15. Wojtusiak, J., Michalski, R. S., Kaufman, K., Pietrzykowski, J.: Multitype Pattern Discovery Via AQ21: A Brief Description of the Method and Its Novel Features. Reports of the Machine Learning and Inference Laboratory, MLI 06-2, George Mason University, Fairfax, VA (2006)

Frequent Pattern Mining and Knowledge Indexing Based on Zero-suppressed BDDs

Shin-ichi Minato and Hiroki Arimura

Graduate School of Information Science and Technology,
Hokkaido University, Sapporo, 060-0814 Japan.

Abstract. Frequent pattern mining is one of the fundamental techniques for knowledge discovery and data mining. In the last decade, a number of efficient algorithms for frequent pattern mining have been presented, but most of them focused on just enumerating the patterns which satisfy the given conditions, and it was a different matter how to store and index the result of patterns for efficient inductive analysis. In this paper, we propose a fast algorithm of extracting all/maximal frequent patterns from transaction databases and simultaneously indexing the result of huge patterns using Zero-suppressed BDDs (ZBDDs). Our method is fast as competitive as the existing state-of-the-art algorithms, and not only enumerating/listing the patterns but also indexing the output data compactly on main memory. After mining, the result of patterns can efficiently be analyzed by using algebraic operations. The data structures of BDDs have already been used in VLSI logic design systems successively, but our method will be the first practical work of applying the BDD-based techniques for data mining area.

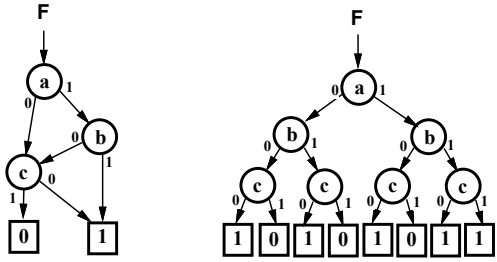
1 Introduction

Frequent pattern mining is one of the fundamental techniques for knowledge discovery and data mining. Since the introduction by Agrawal et al.[1], the frequent pattern mining and association rule analysis have been received much attentions from many researchers, and a number of papers have been published about the new algorithms or improvements for solving such mining problems[7, 9, 19]. However, most of such pattern mining algorithms focused on just enumerating or listing the patterns which satisfy the given conditions and it was a different matter how to store and index the result of patterns for efficient inductive data analysis.

In this paper, we propose a fast algorithm of extracting all/maximal frequent patterns from transaction databases, and simultaneously indexing the result of huge patterns on the computer memory using Zero-suppressed BDDs (ZBDDs). Our method does not only enumerate/list the patterns but also indexes the output data compactly on main memory. After mining, the result of patterns can efficiently be analyzed by using algebraic operations.

The key of our method is to use BDD (Binary Decision Diagrams) -based data structure for representing sets of patterns. BDDs[4] are graph-based representation of Boolean functions, now widely used in VLSI logic design and verification area. For the data mining applications, it is important to use Zero-suppressed BDDs (ZBDDs)[12], a special type of BDDs, which are suitable for handling large-scale sets of combinations. Using ZBDDs, we can implicitly enumerate combinatorial item set data and efficiently compute set operations over the ZBDDs. The preliminary idea of using ZBDDs is presented in our last workshop paper[15], and after that we developed a fast pattern mining algorithm based on this data structure. Our work will be the first practical result of applying the BDD-based technique for data mining area.

For a related work, *FP-tree*[9] receives a great deal of attention because it supports fast manipulation of large-scale item set data using compact tree structure on the main



(a) BDD. (b) Binary tree.

Fig. 1. BDD and binary tree: $F = (a \wedge b) \vee \overline{c}$.

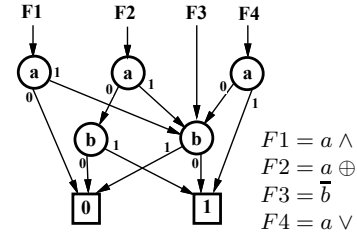


Fig. 2. Shared multiple BDDs.

memory. Our method is a similar approach to handle sets of combinations on the main memory, but will be more efficient in the following points:

- ZBDDs are a kind of DAGs for representing item sets, while FP-tree is a tree representation. In general, DAGs can be more compact than trees.
- Our method uses ZBDDs not only as internal data structure but also as output data structure. It provides an efficient knowledge index for consequent inductive analysis.

Our mining algorithm is based on a recursive depth-first search of the database represented by ZBDDs. We show two versions of algorithms, generating all frequent patterns and generating maximal ones. Experimental result shows that our method is fast as competitive as the existing state-of-the-art algorithms, such as ones based on FP-trees. Especially for the cases where the ZBDD nodes are well shared, exponential speed up are observed comparing to the existing algorithms based on explicit table/tree representation.

Recently, the data mining methods are often discussed in the context of Inductive Databases[3, 11], the integrated processes of knowledge discovery. In this paper, we also show a number of examples of the post processing after frequent pattern mining. We place the ZBDD-based method as a basis of integrated discovery processes to efficiently execute various operations finding interest patterns and analyzing information involved in large-scale combinatorial item set databases.

2 BDDs and Zero-suppressed BDDs

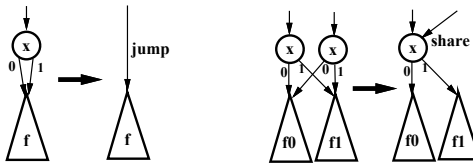
Here we briefly describe the basic techniques of BDDs and Zero-suppressed BDDs for representing sets of combinations efficiently.

2.1 BDDs

BDD (Binary Decision Diagram) is a directed graph representation of the Boolean function, as illustrated in Fig. 1(a). It is derived by reducing a binary tree graph representing recursive *Shannon's expansion*, indicated in Fig. 1(b). The following reduction rules yield a *Reduced Ordered BDD (ROBDD)*, which can efficiently represent the Boolean function. (see [4] for details.)

- Delete all redundant nodes whose two edges point to the same node. (Fig. 3(a))
- Share all equivalent sub-graphs. (Fig. 3(b))

ROBDDs provide canonical forms for Boolean functions when the variable order is fixed. Most researches on BDDs are based on the above reduction rules. In the following sections, ROBDDs will be referred to as BDDs (or ordinary BDDs) for the sake of simplification.



(a) Node deletion. (b) Node sharing.

Fig. 3. Reduction rules of ordinary BDDs

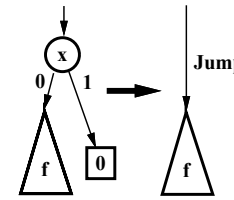


Fig. 4. ZBDD reduction rule.

As shown in Fig. 2, a set of multiple BDDs can share thier subgraphs each other under the same fixed variable ordering. In this way, we can handle a number of Boolean functions simultaneously in a monolithic memory space.

Using BDDs, we can uniquely and compactly represent many practical Boolean functions including AND, OR, parity, and arithmetic adder functions. Using Bryant's algorithm[4], we can efficiently construct a BDD for the result of a binary logic operation (i.e. AND, OR, XOR), for given a pair of operand BDDs. This algorithm is based on hash table techniques, and the computation time is almost linear to the data size unless the data overflows the main memory. (see [13] for details.)

Based on these techniques, a number of BDD packages have been developed in 1990's and widely used for large-scale Boolean function manipulation, especially popular in VLSI CAD area.

2.2 Sets of Combinations and ZBDDs

BDDs are originally developed for handling Boolean function data, however, they can also be used for implicit representation of sets of combinations. Here we call "sets of combinations" for a set of elements each of which is a combination out of n items. This data model often appears in real-life problems, such as combinations of switching devices(ON/OFF), fault combinations, and sets of paths in the networks.

A combination of n items can be represented by an n-bit binary vector, $(x_1 x_2 \dots x_n)$, where each bit, $x_k \in \{1, 0\}$, expresses whether or not the item is included in the combination. A set of combinations can be represented by a list of the combination vectors. In other words, a set of combinations is a subset of the power set of n items.

A set of combinations can be mapped into Boolean space by using n-input variables for each bit of the combination vector. If we choose any one combination vector, a Boolean function determines whether the combination is included in the set of combinations. Such Boolean functions are called *characteristic functions*. For example, the left side of Fig. 5 shows a truth-table representing a Boolean function $(ab\overline{c}) \vee (\overline{b}c)$, but also represents a set of combination $\{ab, ac, c\}$. Using BDDs for characteristic functions, we can implicitly and compactly represent sets of combinations. The logic operations AND/OR for Boolean functions correspond to the set operations intersection/union for sets of combinations. By using BDDs for characteristic functions, we can manipulate sets of combinations efficiently. They can be generated and manipulated within a time roughly proportional to the BDD size. When we handle many combinations including similar patterns (sub-combinations), BDDs are greatly reduced by node sharing effect, and sometimes an exponential reduction benefit can be obtained.

Zero-suppressed BDD (ZBDD)[12, 14] is a special type of BDDs for efficient manipulation of sets of combinations. ZBDDs are based on the following special reduction rules.

- Delete all nodes whose 1-edge directly points to the 0-terminal node, and jump through to the 0-edge's destination, as shown in Fig. 4.
- Share equivalent nodes, similarly to ordinary BDDs.

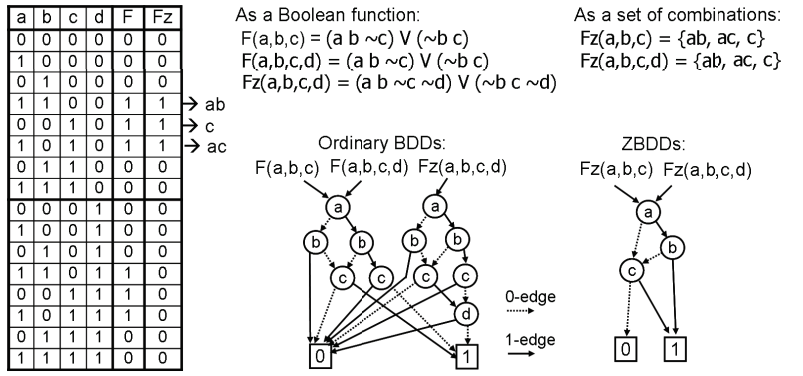


Fig. 5. Effect of ZBDD reduction rule.

Notice that we do not delete the nodes whose two edges point to the same node, which used to be deleted by the original rule. The zero-suppressed deletion rule is asymmetric for the two edges, as we do not delete the nodes whose 0-edge points to a terminal node. It is proved that ZBDDs are also gives canonical forms as well as ordinary BDDs under a fixed variable ordering.

Here we summarize the features of ZBDDs.

- In ZBDDs, the nodes of irrelevant items (never chosen in any combination) are automatically deleted by ZBDD reduction rule. In ordinary BDDs, irrelevant nodes still remain and they may spoil the reduction benefit of sharing nodes. An example is shown in Fig. 5. In this case, the item d is irrelevant, but ordinary BDD for characteristic function $Fz(a, b, c)$ and $Fz(a, b, c, d)$ become different forms. On the other hand, ZBDDs for $Fz(a, b, c)$ and $Fz(a, b, c, d)$ become identical forms and completely shared.
- Each path from the root node to the 1-terminal node corresponds to each combination in the set. Namely, the number of such paths in the ZBDD equals to the number of combinations in the set. In ordinary BDDs, this property does not always hold.
- When no equivalent nodes exist in a ZBDD, that is the worst case, the ZBDD structure explicitly stores all items in all combinations, as well as using an explicit linear linked list data structure. Namely, (the order of) ZBDD size never exceeds the explicit representation. If more nodes are shared, the ZBDD is more compact than linear list.

Table 1 shows the most of primitive operations of ZBDDs. In these operations, $\emptyset$, **1**, *P.top* are executed in a constant time, and the others are almost linear to the size of graph. We can describe various processing on sets of combinations by composing of these primitive operations.

2.3 ZBDD-based Database Analysis

In this paper, we discuss the method of manipulating large-scale transaction databases using ZBDDs. Here we consider binary item set databases, each record of which holds a combination of items chosen from a given item list. Such a combination is called a *tuple* (or a *transaction*).

For analyzing those large-scale transaction databases, frequent pattern mining[2] and *maximum frequent pattern mining*[5] are especially important and they have been discussed actively in the last decade. Since the introduction by Agrawal et al.[1], a number of papers have been published about the new algorithms or improvements for solving such mining problems[7, 9, 19]. Recently, graph-based methods, such as

Table 1. Primitive ZBDD operations

"$\emptyset$"	Returns empty set. (0-termial node)
"**1**"	Returns the set of only null-combination. (1-terminal node)
P.top	Returns the item-ID at the root node of P.
P.offset(v)	Subset of combinations not including item v.
P.onset(v)	Gets $P - P$.offset(v) and then deletes v from each combination.
P.change(v)	Inverts existence of v (add / delete) on each combination.
$P \cup Q$	Returns union set.
$P \cap Q$	Returns intersection set.
$P - Q$	Returns difference set. (in P but not in Q.)
P.count	Counts number of combinations.

Table 2. Statistics of typical benchmark data.

Data name	$\#I$	$\#T$	$total\|T\|$	$avg\|T\|$	$avg\|T\|/\#I$
T10I4D100K	870	100,000	1,010,228	10.1	1.16%
mushroom	119	8,124	186,852	23.0	19.32%
pumsb	2,113	49,046	3,629,404	74.0	3.50%
BMS-WebView-1	497	59,602	149,639	2.5	0.51%
accidents	468	340,183	11,500,870	33.8	7.22%

FP-growth[9], receive a great deal of attention, since they can quickly manipulate large-scale tuple data by constructing compact graph structure on the main memory.

ZBDD-based method is a similar approach to handle sets of combinations on the main memory, but will be more efficient because ZBDDs are a kind of DAGs for representing item sets, while FP-growth uses a tree representation for the same objects. In general, DAGs can be more compact than trees.

Another important point is that our method uses ZBDDs not only as internal data structure but also as output data structure. The most of existing state-of-the-art pattern mining algorithms focused on just enumerating or listing the patterns which satisfy the given conditions, and it was a different matter how to store and index the result of patterns for efficient data analysis. In this paper, we present a fast algorithm of pattern mining and simultaneously indexing the result of huge patterns compactly on the main memory for consequent analysis. The results can be analyzed flexibly by using algebraic operations implemented on ZBDDs.

In addition, we show here why we use ZBDDs instead of ordinary BDDs in this application. Table 2 lists the basic statistics of typical data mining benchmark data[7]. $\#I$ shows the number of items used in the data, $\#T$ is the number of tuples included in the data, $avg|T|$ is the average number of items per tuple, and $avg|T|/\#I$ is the average appearance ratio of each item. From this table, we can observe that the item's appearance ratio is very small in many cases. This observation means that we often handle very sparse combinations in many practical data mining/analysis problems, and in such cases, the ZBDD reduction rule is extremely effective. If the average appearance ratio of each item is 1%, ZBDDs are possibly more compact than ordinary BDDs, up to 100 times. In the literature, there is a first report by Jiang et al.[10] applying BDDs to data mining problems, but the result seems not excellent due to the overhead of ordinary BDDs. We must use ZBDDs in stead of ordinary BDDs for success in many practical data mining/analysis problems.

3 ZBDD-based pattern mining algorithm

In this section, we describe our new algorithm, ZBDD-growth, which extract all frequent patterns from a given transaction database using ZBDDs.

3.1 Tuple-Histograms and ZBDD vectors

A *Tuple-histogram* is the table for counting the number of appearance of each tuple in the given database. An example of tuple-histogram is shown in Fig. 6. This is just a compressed table of the database to combine the same tuples appearing more than once into one line with the frequency.

Record ID	Tuple
1	a b c
2	a b
3	a b c
4	b c
5	a b
6	a b c
7	c
8	a b c
9	a b c
10	a b
11	b c

Original database

Tuple	Freq.
a b c	5
a b	3
b c	2
c	1

Tuple-histogram

Fig. 6. Example of tuple-histogram.

tuple	frequency	F_2	F_1	F_0
abc	5 (101)	1	0	1
ab	3 (011)	0	1	1
bc	2 (010)	0	1	0
c	1 (001)	0	0	1

F_0 = {abc, ab, c}
F_1 = {ab, bc}, F_2 = {abc}

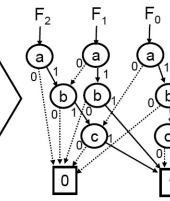


Fig. 7. ZBDD vector for tuple-histogram.

Our pattern mining algorithm manipulates ZBDD-based tuple-histogram representation as the internal data structure. Here we describe how to represent tuple-histograms using ZBDDs. Since ZBDDs are representation of sets of combinations, a simple ZBDD distinguishes only existence of each tuple in the database. In order to represent the numbers of tuple's appearances, we decompose the number into m-digits of ZBDD vector $\{F_0, F_1, \ldots, F_{m-1}\}$ to represent integers up to $(2^m - 1)$, as shown in Fig. 7. Namely, we encode the appearance numbers into binary digital code, as F_0 represents a set of tuples appearing odd times (LSB = 1), F_1 represents a set of tuples whose appearance number's second lowest bit is 1, and similar way we define the set of each digit up to F_{m-1}.

In the example of Fig. 7, The tuple frequencies are decomposed as: $F_0 = \{abc, ab, c\}$, $F_1 = \{ab, bc\}$, $F_2 = \{abc\}$, and then each digit can be represented by a simple ZBDD. The three ZBDDs share their sub-graphs each other.

Now we explain the procedure for constructing a ZBDD-based tuple-histogram from original database. We read a tuple data one by one from the database, and accumulate the single tuple data to the histogram. More concretely, we generate a ZBDD of T for a single tuple picked up from the database, and accumulate it to the ZBDD vector. The ZBDD of T can be obtained by starting from "**1**" (a null-combination), and applying "Change" operations several times to join the items in the tuple. Next, we compare T and F_0, and if they have no common parts, we just add T to F_0. If F_0 already contains T, we eliminate T from F_0 and carry up T to F_1. This ripple carry procedure continues until T and F_k have no common part. After finishing accumulations for all data records, the tuple-histogram is completed.

Using the notation F.add(T) for addition of a tuple T to the ZBDD vector F, we describe the procedure of generating tuple-histogram H for given database D.

$H = \mathbf{0}$
forall $T \in D$ **do**
  $H = H$.add(T)
return H

When we construct a ZBDD vector of tuple-histogram, the number of ZBDD nodes in each digit is bounded by total appearance of items in all tuples. If there are many partially similar tuples in the database, the sub-graphs of ZBDDs are shared very well, and compact representation is obtained. The bit-width of ZBDD vector is bounded by $\log S_{max}$, where S_{max} is the appearance of most frequent items.

Once we have generated a ZBDD vector for the tuple-histogram, various operations can be executed efficiently. Here are the instances of operations used in our pattern mining algorithm.

- H.factor0(v): Extracts sub-histogram of tuples without item v.
- H.factor1(v): Extracts sub-histogram of tuples including item v and then delete v from the tuple combinations. (also considered as the quotient of H/v)

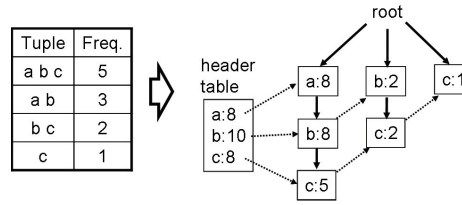


Fig. 8. Example of FP-tree.

```
ZBDDgrowth(H, α)
{
  if(H has only one item v)
    if(v appears more than α )
      return v ;
    else return "0" ;
  F ← Cache(H) ;
  if(F exists) return F ;
  v ← H.top ; /* Top item in H */
  H1 ← H.factor1(v) ;
  H0 ← H.factor0(v) ;
  F1 ←ZBDDgrowth(H1, α) ;
  F0 ←ZBDDgrowth(H0 + H1, α) ;
  F ← (v · F1) ∪ F0 ;
  Cache(H) ← F ;
  return F ;
}
```

Fig. 9. ZBDD-growth algorithm.

```
ZBDDgrowthMax(H, α)
{
  if(H has only one item v)
    if(v appears more than α )
      return v ;
    else return "0" ;
  F ← Cache(H) ;
  if(F exists) return F ;
  v ← H.top ; /* Top item in H */
  H1 ← H.factor1(v) ;
  H0 ← H.factor0(v) ;
  F1 ←ZBDDgrowthMax(H1, α) ;
  F0 ←ZBDDgrowthMax(H0 + H1, α) ;
  F ← (v · F1) ∪ (F0 − F0.permit(F1)) ;
  Cache(H) ← F ;
  return F ;
}
```

Fig. 10. ZBDD-growth-max algorithm.

- $v \cdot H$: Attaches an item v on each tuple combinations in the histogram F.
- $H_1 + H_2$: Generates a new tuple-histogram with sum of the frequencies of corresponding tuples.
- H.tuplecount: The number of tuples appearing at least once.

These operations can be composed as a sequence of ZBDD operations. The result is also compactly represented by a ZBDD vector. The computation time is bounded by roughly linear to total ZBDD sizes.

3.2 ZBDD vectors and FP-trees

FP-growth[9], one of the state-of-the-art algorithm, constructs "FP-tree" for a given transaction database, and then searches frequent patterns using this data structure. An example of FP-tree is shown in Fig. 8. We can see that FP-tree is a "trie" of tuples with their frequencies. In other words, **FP-growth is based on the tree representation of tuple-histograms.** Namely, ZBDD-growth is based on logically same internal data structure as FP-growth. This is the reason why we call this algorithm ZBDD-growth. However, ZBDD-based method will be more efficient because ZBDDs can share the equivalent subgraphs and computation time is bounded by the ZBDD size. The benefit of ZBDDs is especially remarkable when a huge number of patterns are produced.

3.3 Frequent Pattern Mining Algorithm

Our algorithm, ZBDD-growth, is based on a recursive depth-first search over the ZBDD-based tuple-histogram representation. The basic algorithm is shown in Fig. 9.

In this algorithm, we choose an item v used in the tuple-histogram H, and compute the two sub-histograms H_1 and H_0. (Namely, $H = (v \cdot H_1) \cup H_0$.) As v is the top item in the ZBDD vector, H_1 and H_0 can be obtained just by referring the 1-edge

```
P.permit(Q)
{
  if(P ="0" or Q ="0") return "0" ;
  if(P = Q) return F ;
  if(P ="1") return "1" ;
  if(Q ="1")
    if(P include "1" ) return "1" ;
    else return "0" ;
  R ← Cache(P, Q) ;
  if(R exists) return R ;
  v ←TopItem(P, Q) ; /* Top item in P, Q */
  (P0, P1) ←factors of P by v ;
  (Q0, Q1) ←factors of Q by v ;
  R ← (v · P1.permit(Q1)) ∪ (P0.permit(Q0 ∪ Q1)) ;
  Cache(P, Q) ← R ;
  return R ;
}
```

Fig. 11. Permit operation.

and 0-edge of the highest ZBDD-node, so the computation time is constant for each digit of ZBDD.

The algorithm consists of the two recursive calls, one of which computes the subset of patterns including v, and the other computes the patterns excluding v. The two subsets of patterns can be obtained as a pair of pointers to ZBDDs, and then the final result of ZBDD is computed. This procedure may require an exponential number of recursive calls, however, we prepare a hash-based cache to store the result of each recursive call. Each entry in the cache is formed as pair (H, F), where H is the pointer to the ZBDD vector for a given tuple-histogram, and F is the pointer to the result of ZBDD. On each recursive call, we check the cache to see whether the same histogram H has already appeared, and if so, we can avoid duplicate processing and return the pointer to F directly. By using this technique, the computation time becomes almost linear to the total ZBDD sizes.

In our implementation, we use some simple techniques to prune the search space. For example, if H_1 and H_0 are equivalent, we may skip to compute F_0. For another case, we can stop the recursive calls if total frequencies in H is no more than α. There are some other elaborate pruning techniques, but they needs additional computation cost for checking the conditions, so sometimes effective but not always.

3.4 Extension for Maximal Pattern Mining

We can extend the ZBDD-growth algorithm to extract only the maximal frequent patterns[5], each of which is not included in any other frequent patterns. The algorithm is shown in Fig. 10.

The difference from the original algorithm is only one line, written in the frame box. In this part, we check each pattern in F_0, and delete it if the pattern is included in one of patterns of F_1. In this way, we can generate only maximal frequent patterns. This is basically the same approach as used in MAFIA[5].

The process of deleting non-maximal patterns is basically a very time consuming task, however, we found that one of the ZBDD-based operation, called *permit* operation by Okuno et al.[17], can be used for solving this problem[1]. P.permit(Q) returns a set of combinations in P each of which is a subset of some combinations in Q. For example, when $P = \{ab, abc, bcd\}$ and $Q = \{abc, bc\}$, then P.permit(Q) returns $\{ab, abc\}$. The permit operation is efficiently implemented as a recursive procedure of ZBDD manipulation, as shown in Fig. 3.4. The computation time of permit operation is almost linear to the ZBDD size.

[1] Permit operation is basically same as *SubSet* operation by Coudert et al.[6], defined for ordinary BDDs.

Table 3. "One-pair-missing."

	$a_2b_2a_3b_3\cdots a_{n-1}b_{n-1}a_nb_n$		
a_1b_1	$a_3b_3\cdots a_{n-1}b_{n-1}a_nb_n$		
$a_1b_1a_2b_2$	$\cdots a_{n-1}b_{n-1}a_nb_n$		
$\vdots$	$\ddots$	$\vdots$	
$a_1b_1a_2b_2a_3b_3\cdots$		a_nb_n	
$a_1b_1a_2b_2a_3b_3\cdots a_{n-1}b_{n-1}$			

Table 4. Results for "one-pair-missing."

n	#Patterns	(output) |ZBDD|	ZBDD-growth Time(sec)	FP-growth Time(sec)
8	58,974	35	0.01	0.11
10	989,526	45	0.01	1.93
12	16,245,774	55	0.01	32.20
14	263,652,486	65	0.02	518.90
15	1,059,392,916	70	0.02	1966.53
16	4,251,920,574	75	0.02	(timeout)

Table 5. Generation of tuple-histograms.

Data name	$\#T$	$total\|T\|$	|ZBDD Vector|	Time(s)
T10I4D100K	100,000	1,010,228	552,429	43.2
mushroom	8,124	186,852	8,006	1.2
pumsb	49,046	3,629,404	1,750,883	188.5
BMS-WebView-1	59,602	149,639	46,148	18.3
accidents	340,183	11,500,870	3,877,333	107.0

4 Experimental Results

Here we show the experimental results to evaluate our new method. We used a Pentium-4 PC, 800MHz, 1.5GB of main memory, with SuSE Linux 9. We can deal with up to 20,000,000 nodes of ZBDDs in this machine.

4.1 Experiment for Mathematical Example

First, we present the experiment for a set of artificial examples where ZBDD-growth is extremely effective. The database, named "one-pair-missing," has the form as shown in Table 3. Namely, this database has n records each of which contains $(n-1)$ pairs of items but only one pair is missing. It may produce an exponential number of frequent patterns. The experimental results with frequency threshold $\alpha = 1$ are shown in Table 4. We can observe the exponential explosion of the number of patterns, but only linear size of ZBDDs are needed for representing such a huge number of patterns. In such cases, ZBDD-growth runs extremely fast, while FP-growth requires exponential time depending on the output data size.

4.2 Experiments for Benchmark Examples

Next we show the results for the benchmark examples[8], written in previous section.

Table 5 shows the time and space for generating ZBDD vectors of tuple-histograms. In this table, $\#T$ shows the number of tuples, $total|T|$ is the total of tuple sizes (total appearances of items), and |ZBDD| is the number of ZBDD nodes for the tuple-histograms. We can see that tuple-histograms can be constructed for all instances in a feasible time and space. The ZBDD sizes are almost same or less than $total|T|$.

After generating ZBDD vectors for the tuple-histograms, we applied ZBDD-growth algorithm to generate frequent patterns. Table 6 show the results for the selected benchmark examples, "mushroom," "T10I4D100K," and "BMS-WebView-1." The execution time includes the time for generating the initial ZBDD vectors for tuple-histograms.

The results shows that ZBDD-growth is much faster than FP-tree for "mushroom," but not effective for "T10I4D100K." "T10I4D100K" is known as an artificial database, consists of randomly generated combinations, so there are almost no relationship between the tuples. In such cases, ZBDD nodes cannot be shared well, and only the overhead factor is revealed. For "BMS-WebView-1," ZBDD-growth is slower than FP-growth when the output size is small, however, an exponential factor of reduction is observed for the cases of generating huge patterns. Especially for $\alpha = 31, 30$, more than 1 Tera patterns are generated and compactly stored in the memory, that has never been possible by using conventional data structures.

Table 6. Results for benchmark examples.

Data name: Min. freq. α	#Frequent patterns	(output) |ZBDD|	ZBDD-growth Time(s)	FP-growth Time(s)
mushroom: 5,000	41	11	1.2	0.1
1,000	123,277	1,417	3.7	0.3
200	18,094,821	12,340	9.7	5.4
16	1,176,182,553	53,804	7.7	244.1
4	3,786,792,695	59,970	4.3	891.3
1	5,574,930,437	40,557	1.8	1,322.5
T10I4D100K: 5,000	10	10	81.3	0.7
1,000	385	382	135.5	3.1
200	13,255	4,288	279.4	4.5
16	175,915	89,423	543.3	13.7
4	3,159,067	1,108,723	646.0	38.8
1	2,217,324,767	(mem.out)	–	317.1
BMS-WebView1: 1,000	31	31	27.8	0.2
200	372	309	31.3	0.4
50	8,191	3,753	49.0	0.8
34	4,849,465	64,601	120.8	8.3
32	1,531,980,297	97,692	133.7	345.3
31	8,796,564,756,112	117,101	138.1	(timeout)
30	35,349,566,550,691	152,431	143.9	(timeout)

Table 7. Results of maximal pattern mining.

Data name: Min. freq. α	#Maximal freq. patterns	(output) |ZBDD|	ZBDD-growth-max Time(s)	$Time_{(max)}$ $/Time_{(all)}$
mushroom: 5,000	3	10	1.2	1.00
1,000	467	744	4.1	1.10
200	3,111	4,173	10.7	1.10
16	24,060	13,121	8.1	1.06
4	39,456	14,051	4.2	0.98
1	8,124	8,006	1.2	0.70
T10I4D100K: 5,000	10	10	107.1	1.32
1,000	370	376	203.1	1.50
200	1,938	2,609	462.8	1.66
16	68,096	66,274	922.4	1.70
4	400,730	372,993	1141.2	1.77
1	77,443	532,061	140.5	–
BMS-WebView1: 1,000	29	30	34.9	1.25
200	264	289	41.2	1.32
50	3,546	3,064	71.2	1.45
34	15,877	16,854	173.1	1.43
32	15,252	17,680	196.6	1.47
31	13,639	17,383	208.7	1.51
30	11,371	16,323	219.7	1.53

4.3 Maximal Frequent Pattern Mining

We also show the experimental results of maximal frequent pattern mining using ZBDD-growth-max algorithm. In Table 7, we show the results for the same examples as used in the experiment of original ZBDD-growth. The last column $Time_{(max)}/Time_{(all)}$ shows the ratio of computation time between the ZBDD-growth-max and the original ZBDD-growth algorithm. We can observe that the computation time is almost the same (up to twice) between the two algorithms. In other words, the additional computation cost for ZBDD-growth-max is almost the same order as the original algorithm. Our ZBDD-based "permit" operation can efficiently filter the maximal patterns within a time depending on the ZBDD size, which is almost the same cost as manipulating ZBDD vectors of tuple-histograms.

5 Post Processing for Generated Frequent Patterns

Our ZBDD-based method features that the algorithm uses ZBDDs not only as internal data structure but also as output data structure indexing a huge number of patterns compactly on the main memory. The results can be analyzed flexibly by using algebraic operations implemented on ZBDDs. Here we show several examples of the post processing operations for the output data.

(Sub-pattern matching for the frequent patterns): For the result of frequent patterns F, we can efficiently filter a subset S, such that each pattern in S contains a given sub-pattern P.

$S = F$
forall $v \in P$ **do:**
  $S = S.\text{onset}(v).\text{change}(v)$
return S

Inversely, we can extract a subset of patterns not satisfying the given conditions. It is easily done by computing $F - S$. The computation time for the sub-pattern matching is much smaller than the time for frequent pattern mining.

The above operations are sometimes called constraint pattern mining. In conventional method, it is too time consuming to generate all frequent patterns before filtering. Therefore, many researchers consider the direct methods of constraint pattern mining without generating all patterns. However, using ZBDD-based method, a huge number of patterns can be stored and indexed compactly on main memory, so in many cases, it is possible to generate all frequent patterns and then processing them using algebraic ZBDD operations.

(Extracting Long/Short Patterns): Sometimes we are interested in the long/short patterns, consists of a large/small number of items. Using ZBDDs, all combinations of less than k out of n items are efficiently represented in a polynomial size, bounded by $\mathrm{O}(k \cdot n)$. This ZBDD represents a length constraint of patterns. We then apply intersection (or difference) operation to the frequent patterns with the length constraint of ZBDD. In this way, we can easily extract a set of long/short frequent patterns.

(Comparison of Two Sets of Frequent Patterns): Our ZBDD manipulation environment can efficiently store more than one results of frequent pattern mining together. So, we can compare the two sets of frequent patterns generated with different conditions. For example, if the database is gradually changing as time passing, the tuple-histograms and frequent patterns are not the same forever. Our ZBDD-based method can store and index a number of snapshot of pattern sets and easily show the intersection, union, and difference between any pair of snapshots. When many similar ZBDDs are generated, their ZBDD nodes are effectively shared into a monolithic multi-rooted graph, so the memory requeirement is much less than storing each ZBDD separately.

(Calculating Statistical Data): After generating a ZBDD for a set of patterns, we can quickly count a number of patterns by using a primitive ZBDD operation S.count. The computation time is linearly bounded by ZBDD size, not depending on the amount of pattern counts. We can also efficiently calculate other statistical measures, such as *Support* and *Confidence*, which are often used in probabilistic analysis and machine learning.

(Finding Disjoint Decompositions in Frequent Patterns): In the recent paper[16], we presented an efficient ZBDD-based method for finding all possible *simple disjoint decompositions* in a set of combinations. If a given set of patterns f can be decomposed as $f(X, Y) = g(h(X), Y)$, and X and Y has no common items, we then call it a simple disjoint decomposition.

This decomposition method extracts another aspect of hidden structures from complicated itemset data. The decomposition procedure is enough fast for handling large-scale sets of patterns. It will be a powerful tool for database analysis.

6 Conclusion

In this paper, we presented a new method of ZBDD-based frequent pattern mining algorithm. Our method generates a ZBDD for a set frequent patterns from the ZBDD vector for the tuple-histogram of a given transaction database. Our experimental result shows that our ZBDD-growth algorithm is fast as competitive as the existing

state-of-the-art algorithms, such as FP-growth. Especially for the cases where the ZBDD nodes are well shared, exponential speed up are observed comparing to the existing algorithms based on explicit table/tree representation. On the other hand, for the cases where ZBDD nodes are not well shared, or number of patterns is very small, ZBDD-growth method is not effective and the overhead factor reveals.

However, we do not have to use ZBDD-growth algorithm for all instances. We may use the existing methods for the instances where they are more effective than ZBDD-growth. In addition, we can develop a hybrid program that using FP-tree or simple array for internal data structure, but the output is constructed as a ZBDD.

A ZBDD can be regarded as a compressed "trie" for representing a set of patterns. ZBDD-based method will be useful as a fundamental techniques for database analysis and knowledge indexing, and will be utilized for various applications of inductive data analysis.

References

1. R. Agrawal, T. Imielinski, and A. N. Swami, Mining Association rules between sets of items in large databases, In P. Buneman and S. Jajodia, edtors, *Proc. of the 1993 ACM SIGMOD International Conference on Management of Data*, Vol. 22(2) of SIGMOD Record, pp. 207–216, ACM Press, 1993.
2. R. Agrawal, H. Mannila, R. Srikant, H. Toivonen and A. I. Verkamo, Fast Discovery of Association Rules, In *Advances in Knowledge Discovery and Data Mining*, MIT Press, 307–328, 1996.
3. J.-F. Boulicaut, Proc. 2nd International Workshop on Knowledge Discovery in Inductive Databases (KDID'03), Cavtat-Dubrovnik, 2003.
4. Bryant, R. E., Graph-based algorithms for Boolean function manipulation, IEEE Trans. Comput., C-35, 8 (1986), 677–691.
5. D. Burdick, M. Calimlim, J. Gehrke, MAFIA: A Maximal Frequent Itemset Algorithm for Transactional Databases, In Proc. ICDE 2001, 443–452, 2001.
6. O. Coudert, J. C. Madre, H. Fraisse, A new viewpoint on two-level logic minimization, *in Proc. of 30th ACM/IEEE Design Automation Conference*, pp. 625-630, 1993.
7. B. Goethals, "Survey on Frequent Pattern Mining", Manuscript, 2003. `http://www.cs.helsinki.fi/ u/goethals/publications/survey.ps`
8. B. Goethals, M. Javeed Zaki (Eds.), Frequent Itemset Mining Dataset Repository, Frequent Itemset Mining Implementations (FIMI'03), 2003.
   s `http://fimi.cs.helsinki.fi/data/`
9. J. Han, J. Pei, Y. Yin, R. Mao, Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach, Data Mining and Knowledge Discovery, 8(1), 53–87, 2004.
10. L. Jiang, M. Inaba, and H. Imai: A BDD-based Method for Mining Association Rules, in Proceedings of 55th National Convention of IPSJ, Vol. 3, pp. 397-398, Sept. 1997, IPSJ.
11. H. Mannila, H. Toivonen, Multiple Uses of Frequent Sets and Condensed Representations, In *Proc. KDD*, 189–194, 1996.
12. S. Minato: Zero-suppressed BDDs for set manipulation in combinatorial problems, In Proc. 30th ACM/IEEE Design Automation Conf. (DAC-93), (1993), 272–277.
13. S. Minato: "Binary Decision Diagrams and Applications for VLSI CAD", Kluwer Academic Publishers, November 1996.
14. S. Minato, Zero-suppressed BDDs and Their Applications, International Journal on Software Tools for Technology Transfer (STTT), Springer, Vol. 3, No. 2, pp. 156–170, May 2001.
15. S. Minato and H. Arimura: Efficient Combinatorial Item Set Analysis Based on Zero-Suppressed BDDs", In *Proc. of IEEE/IEICE/IPSJ International Workshop on Challenges in Web Information Retrieval and Integration (WIRI-2005)*, pp. 3-10, Apr., 2005.
16. S. Minato: Finding Simple Disjoint Decompositions in Frequent Itemset Data Using Zero-suppressed BDD, In *Proc. of IEEE ICDM 2005 workshop on Computational Intelligence in Data Mining*, pp. 3-11, ISBN-0-9738918-5-8, Nov. 2005.
17. H. Okuno, S. Minato, and H. Isozaki: On the Properties of Combination Set Operations, Information Procssing Letters, Elsevier, 66 (1998), pp. 195-199, 1998.
18. Ricardo Baeza-Yates, Berthier Ribiero-Neto, "Modern Information Retrieval", Addison Wesley, 1999.
19. M. J. Zaki, Scalable Algorithms for Association Mining, IEEE Trans. Knowl. Data Eng. 12(2), 372–390, 2000.

Quantitative Episode Trees*

Mirco Nanni[1] and Christophe Rigotti[1,2]

[1]KDD Laboratory, University of Pisa and ISTI-CNR Pisa, Italy
[2]INSA-LIRIS UMR 5205 CNRS, Lyon, France

Abstract. Among the family of the local patterns, episodes are commonly used when mining a single or multiple sequences of discrete events. An episode reflects a qualitative relation *is-followed-by* over event types, and the refinement of episodes to incorporate quantitative temporal information is still an on going research, with many application opportunities. In this paper, focusing on serial episodes, we design such a refinement called *quantitative episodes* and give a corresponding extraction algorithm. The three most salient features of these quantitative episodes are: (1) their ability to characterize main groups of homogeneous behaviors among the occurrences, according to the duration of the *is-followed-by* steps, and providing quantitative bounds of these durations organized in a tree structure; (2) the possibility to extract them in a complete way; and (3) to perform such extractions at the cost of a limited overhead with respect to the extraction of standard episodes.

1 Introduction

Sequential data is a common form of information available in several application contexts, thus naturally inducing a strong interest for them among data analysts. A decade-long attention has been paid by researchers in data mining to study forms of patterns appropriated to this kind of data, such as sequential patterns [1] and episodes [7]. In particular, in this paper we will focus on *serial episodes*, that are sequences of event types extracted from single or multiple input sequences, and that reflect a qualitative relation *is-followed-by* between the event types.

Episodes have natural applications into several domains, including for instance the analysis of business time series [2], medical data [8], geophysical data [9] and also alarm log analysis for network monitoring (especially in telecommunications) [5]. However, in many applications episodes clearly show some limitations, due to the fact that the information provided by the *is-followed-by* relation is not always enough to properly characterize the phenomena at hand. This, in particular, pulls our research toward the refinement of episodes to incorporate quantitative temporal information, able to describe the time intervals observed for the *is-followed-by* relation.

* This research is partly funded by EU contracts IQ IST-FP6-516169, and GeoPKDD IST-FP6-014915.

In this paper, we propose a refinement of episodes called *quantitative episodes*, that provides quantitative temporal information in a readable, tree-based graphically representable form. These quantitative episodes describe the main groups of homogeneous behaviors within the occurrences of each episode, according to the elapsed times between the consecutive event types of the episode. Moreover, they are not provided in an isolated way, but in trees giving a global view of how the occurrences of the corresponding episode differentiate in homogeneous groups along the elements of the pattern. From a computational point of view, the main interest of the quantitative episodes is that they can be mined in a sound and complete way without increasing the cost of extractions significantly when compared to extractions of episodes alone. This is achieved through an extraction algorithm that tightly integrates episode extraction with a computationally reasonable analysis of temporal quantitative information.

This paper is organized as follows: in Section 2 some preliminary definitions needed concerning episodes are recalled from the literature; Section 3, then, introduces quantitative episodes; Section 4 presents the principle of an algorithm for efficiently extracting quantitative episodes, which is evaluated experimentally in Section 5; finally, in Section 6 we briefly review the related literature and conclude with a summary in Section 7.

2 Preliminary definitions

We briefly introduce standard notions [7], or give equivalent definitions when more appropriated to our presentation.

Definition 1. ***(event, event sequence,operator $\sqsubseteq$)*** Let E be a set of *event types* and $\prec$ a total order on E. An *event* is a pair denoted (e,t) where $e \in E$ and $t \in \mathbb{N}$. The value t denotes the time stamp at which the event occurs. An *event sequence* S is a tuple of events $S = \langle (e_1,t_1),(e_2,t_2),\ldots,(e_l,t_l)\rangle$ such that $\forall i \in \{1,\ldots,l-1\}$, $t_i < t_{i+1} \vee (t_i = t_{i+1} \wedge e_i \prec e_{i+1})$. Given two sequences of events S and S', S' is a *subsequence* of S, denoted $S' \sqsubseteq S$, if S' is equal to S or if S' can be obtained by removing some elements in S.

Definition 2. ***(episode, occurrence, minimal occurrence, support)*** An *episode* is a non empty tuple α of the form $\alpha = \langle e_1, e_2, \ldots, e_k\rangle$ with $e_i \in E$ for all $i \in \{1,\ldots,k\}$. In this paper, we will use the notation $e_1 \rightarrow e_2 \rightarrow \ldots \rightarrow e_k$ to denote the episode $\langle e_1, e_2, \ldots, e_k\rangle$ where '$\rightarrow$' may be read as 'is followed by'. The *size* of α is denoted $|\alpha|$ and is equal to the number of elements of the tuple α, i.e., $|\alpha| = k$. The *prefix* of α is the episode $\langle e_1, e_2, \ldots, e_{k-1}\rangle$. We denote it as $prefix(\alpha)$. An episode $\alpha = \langle e_1, e_2, \ldots, e_k\rangle$ *occurs* in an event sequence S if there exists at least one sequence of events $S' = \langle (e_1,t_1),(e_2,t_2),\ldots,(e_k,t_k)\rangle$ such that $\forall i \in \{1,\ldots,k-1\}, t_i < t_{i+1}$ and $S' \sqsubseteq S$. The pair $\langle t_1, t_k\rangle$ is called an *occurrence* of α in S. Moreover, if there is no other occurrence $\langle t'_1, t'_k\rangle$ such that $[t'_1, t'_k] \subset [t_1, t_k]$, then the pair $\langle t_1, t_k\rangle$ is called a *minimal occurrence* of α. The support of α in S, denoted $support(\alpha, S)$, is the number of minimal occurrences of α in S.

Intuitively, a minimal occurrence is simply an occurrence that does not strictly contain another occurrence of the same episode. These episodes and their occurrences correspond to the *serial* episodes of [7]. For instance, let $S = \langle (a,0),(b,1),(c,1),(b,2)\rangle$ be an event sequence and $\alpha = a \rightarrow b$ be an episode. Then, α has two occurrences in S: $\langle 0,1\rangle$ and $\langle 0,2\rangle$. The former is a minimal occurrence, while the latter is not, since $[0,1] \subset [0,2]$. Notice that there is no occurrence of episode $\alpha' = b \rightarrow c$.

These definitions, and the ones introduced in the rest of the paper, are given for a single sequence S, but they extend trivially to multiple sequences. In that case the support is the sum of the number of occurrences in all sequences.

3 Quantitative episodes

3.1 Informal presentation

The idea of quantitative episodes essentially consists in dividing the set of occurrences of an episode into homogeneous, significantly populated groups. Homogeneity, in particular, is obtained when on each step, made of two consecutive elements of the episode, the occurrences in the same group show similar transition times (i.e., similar times elapsed between an element and the next one within the episode). The result can be graphically summarized through a tree-like structure, as the one depicted in Figure 1 that represents homogeneous groups of occurrences of an episode $\alpha = A \rightarrow B \rightarrow C \rightarrow D$. The figure can be read in the following way:

- The episode has 1000 occurrences in the sequence of events, and this value is written under the first event of the episode.
- Among these 1000 occurrences, there are 2 subgroups that show homogeneous duration for step $A \rightarrow B$: one (the upper branch of the split) corresponds to transition times between 2 and 10, and covers 500 occurrences; the other (lower branch) corresponds to transition times in interval $[15,20]$ and covers 400 occurrences. Notice that 100 occurrences of $A \rightarrow B \rightarrow C \rightarrow D$ are lost, meaning that they exhibit a rather isolated duration for step $A \rightarrow B$ and cannot be associated with other occurrences to form a significantly populated group.
- In the largest group obtained above, all occurrences present similar step durations for steps $B \rightarrow C$ and $C \rightarrow D$, and are kept together in a single group. The other group, containing 400 occurrences, is split further into homogeneous groups w.r.t. duration of step $B \rightarrow C$. Notice that the resulting homogeneous groups overlap, sharing a subset of occurrences and resulting in non-disjoint time intervals. Indeed, we can observe that the total count of occurrences in the two groups (205+202) is greater than the original total amount (400), since some occurrences are counted twice.

– One of these two groups is further split into two (disjoint) groups while the other is not.
– Each path from the root to a leaf in the tree corresponds to a group of occurrences that shows an homogeneous behavior along all the steps of the episode, and covers a sufficient number of occurrences (in this example, at least 90). This homogeneous behavior can be represented by the sequence of time intervals on the path, and can be added to the episode as a *quantitative* feature to form a *main grouping quantitative episode*. The tree in Figure 1 depicts four such patterns (one for each path from the root to a leaf). The tree relates these patterns together, showing how the occurrences can be differentiated into groups along the steps of the episode.

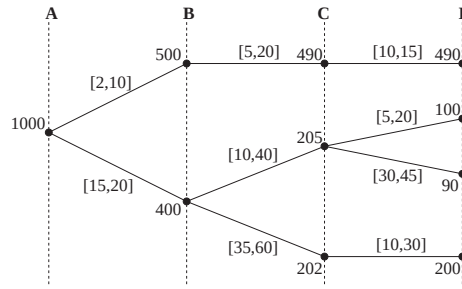


Fig. 1. Tree of quantitative episodes for episode $\alpha = A \rightarrow B \rightarrow C \rightarrow D$.

3.2 Quantitative episode definition

Definition 3. ***(quantitative episode)*** A *quantitative episode* (*q-episode*) is a pair $P = \langle \alpha, IT \rangle$ where α is an episode of size $k > 1$, and $IT = \langle it_1, \ldots, it_{k-1} \rangle$, with $\forall i \in \{1, \ldots, k-1\}, it_i = [a_i, b_i] \subset \mathbb{N}^+$ (i.e., it_i is an interval in $\mathbb{N}^+$). The size of P, denoted $|P|$ is defined as $|P| = |\alpha|$.

The it_i intervals are intended to represent values of elapsed time between the occurrences of two consecutive event types of the episode α. For instance $\langle A \rightarrow B \rightarrow C \rightarrow D, \langle [15, 20], [10, 40], [5,\ 20] \rangle \rangle$ is one of the q-episodes depicted in Figure 1.

To handle the time stamps of the events corresponding to all event types within an episode the definition of occurrence needs to be modified as follows.

Definition 4. ***(occurrence)*** An *occurrence* of an episode $\alpha = \langle e_1, e_2, \ldots, e_k \rangle$ in an event sequence S is a tuple $\langle t_1, t_2, \ldots, t_k \rangle$ such that there exists $S' = \langle (e_1, t_1), (e_2, t_2), \ldots, (e_k, t_k) \rangle$ satisfying $\forall i \in \{1, \ldots, k-1\}, t_i < t_{i+1}$ and $S' \sqsubseteq S$.

Notice that subsequence S' in the definition above can be formed by non-contiguous elements of sequence S. Using this definition of occurrence, the notion of minimal occurrence can be redefined accordingly.

Definition 5. ***(minimal occurrence)*** An occurrence $\langle t_1, \ldots, t_k \rangle$ of an episode α in event sequence S is a *minimal occurrence* if (1) there is no other occurrence $\langle t'_1, \ldots, t'_k \rangle$ of α such that $[t'_1, t'_k] \subset [t_1, t_k]$, and (2) if $k > 2$ then $\langle t_1, \ldots, t_{k-1} \rangle$ is a minimal occurrence of *prefix*(α).

As we will consider only minimal occurrences of episodes, we will simply use the term *occurrence*, when there is no ambiguity.

For a step $e_i \rightarrow e_{i+1}$ in an episode α, and its durations among a set of occurrences of α, now we define how these duration values are grouped. Informally, groups correspond to maximal sets of duration values that form *dense* intervals, where dense means that any sub-interval of significant size w_s contains a significant number of values n_s. More precisely, $w_s \in \mathbb{R}, w_s \geq 1$ and $n_s \in \mathbb{N}^+$ are termed the *density parameters* and characterize the groups in the following definition.

Definition 6. ***(occurrence groups)*** *Let $\mathcal{O}$ be a set of occurrences of episode α and i be an integer parameter such that $1 \leq i < |\alpha|$ (i identifies a step $e_i \rightarrow e_{i+1}$). Let $\Delta_i(x) = t_{i+1} - t_i$ for any occurrence $x = \langle t_1, \ldots, t_{|\alpha|} \rangle$ (i.e., the duration of step $e_i \rightarrow e_{i+1}$ for occurrence x). Then, the* occurrence groups of $\mathcal{O}$ at level i, *denoted as group$(\mathcal{O}, i)$, are defined as follows:*

$$group(\mathcal{O}, i) = \{\ g \mid \begin{array}{l} g \text{ is a maximal subset of } \mathcal{O} \text{ s.t.:} \\ \forall a, b \in [\min_{x \in g} \Delta_i(x), \max_{x \in g} \Delta_i(x)], \\ \quad b - a \geq w_s \ \Rightarrow\ |\{x \in g \mid \Delta_i(x) \in [a, b]\}| \geq n_s \} \end{array}$$

For example, consider the set of occurrences $\mathcal{O} = \{x_1, \ldots, x_8\}$ having the respective durations 3,4,6,6,9,15,16,16 for step $e_i \rightarrow e_{i+1}$ (i.e., the values of Δ_i). Let the density parameters be $w_s = 3$ and $n_s = 2$ (i.e., at least two elements in any sub-interval of size 3). Then $group(\mathcal{O}, i) = \{\{x_1, \ldots, x_5\}, \{x_6, x_7, x_8\}\}$ (corresponding respectively to the durations $3, 4, 6, 6, 9$ and $15, 16, 16$).

The next definition specifies the tree structure of the occurrence groups.

Definition 7. ***(occurrence group tree)*** *Let $\mathcal{O}$ be the set of occurrences of episode α. Then, the* occurrence group tree *(*group tree *for short) of α is a rooted tree with labelled edges such that:*

- *the tree has $|\alpha|$ levels, numbered from 1 (the root) to $|\alpha|$ (the deepest leaves);*
- *each node v is associated with a set $v.g$ of occurrences of α;*
- *the root is associated with $root.g = \mathcal{O}$, i.e., with all the occurrences of α;*
- *if a node v at level i, $1 \leq i < |\alpha|$, is such that $group(v.g, i) = \{g_1, \ldots, g_k\}$, then it has k children $v_1, \ldots, v_k$, with $v_j.g = g_j, i \in \{1, \ldots, k\}$.*
- *each edge connecting node v at level i with its child v_j is labelled with the interval $[\min_{x \in v_j.g} \Delta_i(x), \max_{x \in v_j.g} \Delta_i(x)]$;*

Notice that such tree is unique, up to permutations in the order of the children of each node. Then, the main grouping q-episodes correspond simply to the sets of occurrences that have not been separated from the root to a leaf and that have a significant size.

Definition 8. ***(main grouping q-episode)*** *A q-episode $P = \langle \alpha, IT \rangle$ is said to be* a main grouping q-episode *if the group tree of α contains a path from the root to a leaf v such that:*

- *the labels of the edges met along the path correspond to the intervals in IT;*
- *and $|v.g|$, called the support of P, is greater or equal to σ_g, a user defined minimum group size.*

For instance, Figure 1 depicts a tree of main grouping q-episodes for $\alpha = A \rightarrow B \rightarrow C \rightarrow D$ and $\sigma_g = 90$ (a group tree restricted to paths forming main grouping q-episodes).

Since a minimal occurrence of α can be obtained only by extending a minimal occurrence of $prefix(\alpha)$, we have the following simple property that is used as a safe pruning criterion in the extraction principle.

Theorem 1. *Let α be an episode such that $|\alpha| > 1$. If there exists a main grouping q-episode $\langle \alpha, IT \rangle$, then there exists a main grouping q-episode $\langle prefix(\alpha), IT' \rangle$.*

4 Extracting q-episodes

In this section we present the principles of an algorithm called *Q-epiMiner* to find all main grouping q-episodes. It interleaves frequent episode extraction and group tree computation in a tight and efficient way. A more detailed presentation of the algorithm is given in the report [10].

Let $\alpha = \langle e_1, \ldots, e_n \rangle$ be an episode. For each event type e_i in α, $i > 1$, we consider a list D_i that collects the durations between e_{i-1} and e_i, i.e., the values $\Delta_{i-1}(x)$ for all occurrences x of α, and we suppose that each D_i is sorted by increasing duration value. By convention, for the sake of uniformity, D_1 contains a duration of 0 for all occurrences (there is no element before e_1).

In the following, we describe how these lists $D_1, \ldots, D_n$ can be used to compute the group tree of pattern α, and then how they can be updated when expanding α with an event type e_{n+1}.

Splitting one node. Splitting the group of occurrences of α associated to one node of the tree at level i (to obtain its children at level $i+1$) can be done simply by a single scan of the elements in the group if these elements are ordered by the duration between e_i and e_{i+1}. We use a function named *splitGroup* performing this simple treatment. We suppose that it takes as input a list of occurrences in a group, sorted by duration of $e_i \rightarrow e_{i+1}$, and gives as output a collection of all maximal sublists satisfying the density criterion.

Computing the whole tree. Suppose that we have already computed the groups of occurrences denoted $g_1, \ldots, g_k$ that are associated respectively to the nodes $v_1, \ldots, v_k$ of a level i of the tree. These groups are split in the following way to obtain the nodes of the next level. Firstly, we create for each node v_j an empty

list denoted $v_j.sortedGroup$. Then we scan D_{i+1} from first to last element, and for each occurrence found in D_{i+1} if the occurrence is in a group g_j then we insert the occurrence at the end of $v_j.sortedGroup$. Now, we have at hand for each v_j its group of occurrences sorted by increasing duration between e_i and e_{i+1}. Then, we can apply on each $v_j.sortedGroup$ the *splitGroup* function to compute the children of v_j and their associated groups of occurrences and thus obtain the next level of the group tree. Repeating this process allows to build the group tree in a levelwise way, taking advantage of the sorted lists $D_1, \ldots, D_n$. In the following, we assume that such a tree is computed by a function $computeTree$, applied on a tuple $\langle D_1, \ldots, D_n \rangle$.

Obtaining the information needed to compute the tree. The other key operation is the efficient computation of the sorted lists $D'_1, \ldots, D'_n, D'_{n+1}$ of a pattern $\alpha \rightarrow e$. Suppose that we know the list L_e of occurrences of $\alpha \rightarrow e$, and the sorted lists $D_1, \ldots, D_n$ of occurrences of α. Then, the main property used is that $D'_1, \ldots, D'_n$ are sublists of, respectively, $D_1, \ldots, D_n$, since each occurrence of $\alpha \rightarrow e$ comes from the expansion of an occurrence of α. So a list D'_i can be obtained simply by scanning D_i from the first to the last element and picking (in order) the occurrences in D_i that have been extended to form an occurrence of $\alpha \rightarrow e$. The result is a list D'_i sorted by increasing duration between e_{i-1} and e_i. The case of the list D'_{n+1} is different since it contains the same occurrences as L_e, so D'_{n+1} is simply a copy of L_e, but has to be sorted by increasing duration (between e_n and e_{n+1}). Having at hand the sorted lists $D'_1, \ldots, D'_n, D'_{n+1}$ we can then compute the group tree of $\alpha \rightarrow e$ by calling $computeTree(\langle D'_1, \ldots, D'_n, D'_{n+1} \rangle)$.

Integration with the extraction of episodes. One remaining problem to be solved is to build the occurrence list of the episode under consideration (as the list L_e for $\alpha \rightarrow e$). Fortunately, several approaches to extract episodes, or closely related patterns like *sequential patterns*, are based on the use of such occurrence lists (e.g., [7, 9, 13]), providing the information needed to update the duration lists D_i. Due to space limitation we will not detail this principle here. The basic idea is that if we store in a list L the locations (positions in the data sequence) of the occurrences of a pattern α, then for an event type e, we can use[1] L to build the list L_e of occurrences of $\alpha \rightarrow e$. In our case, for the occurrences of an episode $\alpha = \langle e_1, \ldots, e_n \rangle$ the location information stored in L are simply the time stamps of the last element e_n of α, sorted by increasing value. We use a function *expand* that takes the input sequence S and L, and that returns a set $\mathcal{L}_{exp}$ of tuples $\langle e, L_e \rangle$. The set $\mathcal{L}_{exp}$ contains for each event type e, the list L_e of locations of occurrences of $\alpha \rightarrow e$. As for L, the location information in L_e are the time stamps of the last element of $\alpha \rightarrow e$ and L_e is sorted by increasing location value.

The overall enumeration strategy of the episodes used is a standard depth-first prefix-based strategy because it fits both with the episode extraction and

[1] Together with other information, like the data sequence itself, or the location of the occurrences of e.

with the use of the sorted lists D_i to derive the sorted lists D'_i to compute the group trees. The strategy can simply be sketched as follows: when an episode α is considered we use it as a *prefix* to expand it and to obtain new episodes of the form $\alpha \rightarrow e$, and then, one after the other, we consider and expand each of these $\alpha \rightarrow e$.

Pruning strategy and correctness. Consider an episode α such that all leaves at level $|\alpha|$ of its group tree are associated to groups of size strictly less than σ_g (α has no corresponding main grouping q-episode, but α itself can have a support greater or equal to σ_g). By Theorem 1, we can also safely avoid the expansion of α, since this expansion cannot correspond to any main grouping q-episode. The exhaustive enumeration strategy of the episodes and the safety of the pruning strategy ensure the correctness of the general extraction principle.

5 Experiments

In this section we present the results of a set of experiments mainly aimed at studying how the size of the input data and the value of some input parameters impact on the performances of the *Q-epiMiner* algorithm described in this paper. The experiments presented are made on datasets containing several sequences. As mentioned previously, the definitions extended trivially to that case (the support is simply the sum of the support in all sequences). The only change in the abstract algorithm is that the occurrence locations are not simply time stamps, but sequence identifiers together with time stamps in the sequences. The algorithm was implemented in C, and all experiments were performed on a Intel Xeon 2Ghz processor with 1Gb of RAM over a Linux 2.6.14 platform.

5.1 Performance analysis on synthetic datasets

In order to collect large datasets having controlled characteristics, we randomly generated them by means of the Quest Synthetic Data Generator from IBM[2], by varying the number of input sequences generated (from 10K to 250K), the sequence length[3] (from 5 to 70) and the number of different event types used (from 5K to 20K). Where not specified hereafter, the following default parameter values were adopted: 100K input sequences, sequence length equal to 25, 5K event types, $w_s = 8$ and $n_s = 4$.

The curves in Figure 2(left) show the execution times of the prototype over datasets of increasing size and for three different numbers of event types used in the data. The σ_g parameter was set to 40 for 10K sequences and then was increased proportionally, up to 1000 for 250K sequences. As we can see, the execution time always grows almost linearly, having a higher slope when fewer event

[2] http://www.almaden.ibm.com/software/projects/iis/hdb/Projects/data_mining/mining.shtml

[3] The parameter of the generator controlling the number of events per time stamp was set to 1.

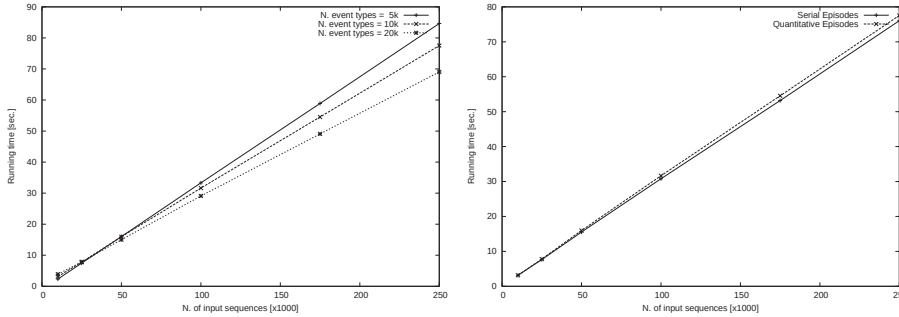


Fig. 2. Scalability w.r.t. number of input sequences.

types are in the data[4]. A similar scalability analysis is provided in Figure 2(right), where *Q-epiMiner* is compared against the extraction of serial episodes having at least a support of σ_g (this extraction is performed using the frequent episodes mining technique embedded in *Q-epiMiner*, without computing the durations, groups and trees, and implemented with the same low level optimizations). The values of σ_g were the same as in the previous experiment. The two curves are very close, meaning that the overhead introduced by the computation of main grouping q-episodes is well balanced by the pruning it allows. Finally, similar results are obtained by varying the length of the input sequences (see Figure 3(left)), where both curves have an apparently-quadratic growth (σ_g was set to 80 for length 5 and then was increased proportionally, up to 1120 for length 70). Obviously, for very long sequences usual episode constraints, like maximum window size, might be used [7].

Figure 3(right) reports the behaviour of the prototype when the minimum size of the groups is varied from 100 to 2000, and again its comparison to the mining of frequent serial episodes at minimum support σ_g. Here also, the two algorithms behave very similarly, this time showing a fast drop in the execution time as σ_g grows – as usual for frequent pattern mining algorithms. Due to space limits, we do not report the results obtained by varying the density parameters, that, however, seemed to have only a very limited impact on execution times of the algorithm on this kind of data.

5.2 Experiments on a real dataset

In this set of experiments we used real world data consisting of the July 2000 weblog from the web server of the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley[5]. In a preprocessing step, all non-HTML pages where removed and user sessions were extracted,

[4] Fewer event types with the same number of sequences leads to higher supports for the remaining event types and more frequent patterns of large size.

[5] http://www.cs.berkeley.edu/logs/http

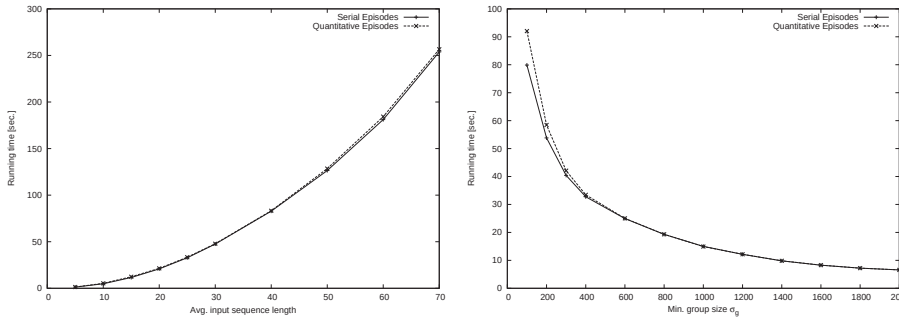


Fig. 3. Scalability w.r.t. input sequence length and min. group size σ_g, with 100K sequences.

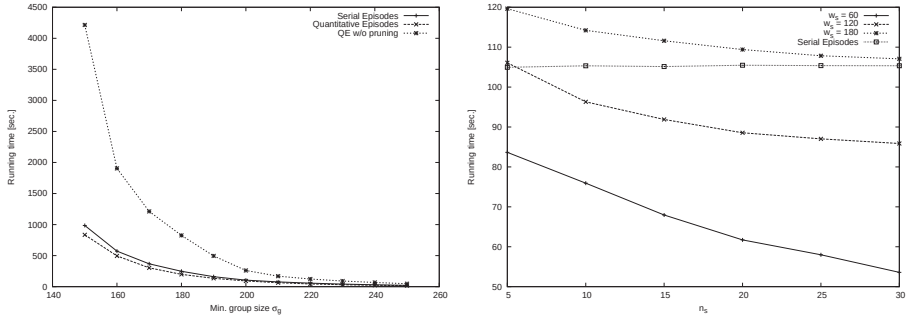


Fig. 4. Berkely dataset: Scalability w.r.t. σ_g and effects of the density parameters.

resulting in 90295 user sessions (used as input sequences) of average length of 13.0 with 72014 distinct pages.

In Figure 4 two graphs are plotted that describe the performances of the *Q-epiMiner* prototype on the Berkeley dataset for different minimum group sizes (graph on the left, with $w_s = 120$ and $n_s = 15$) and different density parameters (on the right, with $\sigma_g = 200$). The first plot confirms the results obtained on synthetic data, i.e., execution times drop very quickly as σ_g decreases. Moreover, an additional curve is plotted that represents a version of *Q-epiMiner* that does not apply any pruning based on the absence of a main grouping q-episode, but only applies a pruning based on the support of the episodes (an episode is not expanded only when its support is strictly less than σ_g). This curve shows the effectiveness of the full pruning made by *Q-epiMiner*. It should also be noticed that on this dataset, *Q-epiMiner* performs even better than the serial episode miner (with minimum support set to σ_g), confirming the fact that the pruning capabilities of the prototype are able to balance its potential overhead.

Finally, Figure 4(right) shows that, quite reasonably, the execution time decreases with larger minimum density parameter n_s (since they allow a stronger

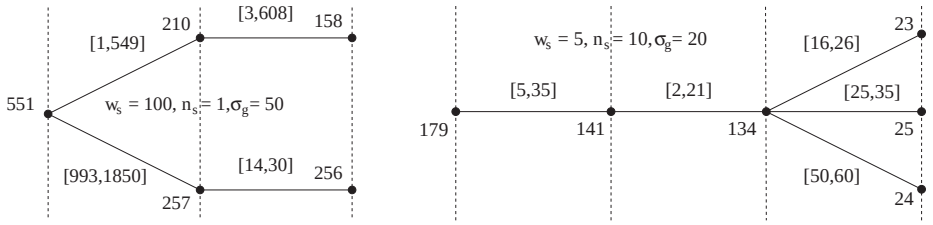


Fig. 5. Examples of trees of main grouping q-episodes.

pruning), and increases with larger window sizes w_s (which acts in the opposite direction).

We conclude this section by providing in Figure 5 two sample outputs obtained from the Berkeley dataset. In particular, we notice that the first tree contains two groups that split at the first step, showing well separated intervals of times ($[1, 549]$ against $[993, 1850]$). On the contrary, the second one contains three groups that split only at the third step, two of which overlap ($[16, 26]$ and $[25, 35]$). In both cases, each time a group splits some of the occurrences it contains are lost, i.e., they are not part of any subgroup (of size at least σ_g) created by the split.

6 Related work

The need of quantitative temporal information in patterns over event sequences has been pointed in recent works in the data mining literature [12, 3, 11, 4, 6, 9].

An important difference between these approaches and the q-episodes introduced here, is that the former provide patterns in isolation, while q-episodes are related in tree structures. Such trees give a global view of how the occurrences of a pattern differentiate in homogeneous groups along the sequence of event types (from the first to the last element of the pattern).

Different notions of intervals are also considered. In [6] the intervals are not determined by the data but are fixed by the user; only the interval between the beginning and the end of a pattern is considered in [9]; and in [3] intervals are derived from intervals of occurrences of patterns of size two only.

The other approaches [12, 11, 4] compute the intervals from the data and for all pattern lengths, as in the case of the q-episodes. However, among these approaches, only [4] considers an exhaustive extraction (at the cost of intrinsically expensive algorithmic solutions), while the others compute only *some* of the patterns using heuristics and/or non-deterministic choices.

Finally, it should be noticed that the overhead of computing the quantitative temporal information was not assessed in these previous works.

7 Conclusion

In this paper we introduced *quantitative episodes*, an extension of serial episodes that refines standard episodes by integrating quantitative temporal information. A tight integration of episode extraction and group tree computation allowed to obtain a complete and efficient algorithm that adds a negligible overhead to the extraction of serial episodes, as assessed by the experimental results on performances. We think that these features, and the possibility of an easy-to-grasp representation of the output into a graphical tree-like structure, make the approach suitable for many applications.

References

1. R. Agrawal and R. Srikant. Mining sequential patterns. In *Proc. of ICDE*, pages 3–14, 1995.
2. G. Das, L. K.I., H. Mannila, G. Renganathan, and P. Padhraic Smyth. Rule discovery from time series. In *Proc. of KDD*, pages 16–22, 1998.
3. C. Dousson and T. V. Duong. Discovering chronicles with numerical time constraints from alarm logs for monitoring dynamic systems. In *Proc. of IJCAI*, pages 620–626, 1999.
4. F. Giannotti, M. Nanni, and D. Pedreschi. Efficient mining of temporally annotated sequences. In *Proc. of the SIAM Conference on Data Mining (SDM)*, 2006.
5. K. Hatonen, M. Klemettinen, H. Mannila, P. Ronkainen, and H. Toivonen. TASA: Telecomunications alarm sequence analyzer or: How to enjoy faults in your network. In *Proc. of IEEE Network Operations and Management Symposium*, pages 520–529, 1996.
6. Y. Hirate and H. Yamana. Sequential pattern mining with time intervals. In *Proc. of PAKDD*, 2006.
7. H. Mannila, H. Toivonen, and A. Verkamo. Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery*, 1(3):259–298, November 1997.
8. N. Meger, C. Leschi, N. Lucas, and C. Rigotti. Mining episode rules in STULONG dataset. In *Proc. of the ECML/PKDD Discovery Challenge*, 2004.
9. N. Meger and C. Rigotti. Constraint-based mining of episode rules and optimal window sizes. In *Proc. of PKDD*, pages 313–324, 2004.
10. M. Nanni and C. Rigotti. Quantitative episode trees. Technical report, 17 pages, 2006.
11. A. Vautier, M.-O. Cordier, and R. Quiniou. An inductive database for mining temporal patterns in event sequences. In *Proc. of ECML/PKDD Workshop on Mining Spatial and Temporal Data*, 2005.
12. M. Yoshida et al. Mining sequential patterns including time intervals. In *Proc. of SPIE Conference on Data Mining and Knowledge Discovery: Theory, Tools and Technology II*, 2000.
13. M. Zaki. Spade: an efficient algorithm for mining frequent sequences. *Machine Learning, Special issue on Unsupervised Learning*, 42(1/2):31–60, Jan/Feb 2001.

IQL: A Proposal for an Inductive Query Language

Siegfried Nijssen and Luc De Raedt

Institut für Informatik, Albert-Ludwigs-Universität,
Georges-Köhler-Allee, Gebäude 097, D-79110, Freiburg im Breisgau, Germany.
snijssen@informatik.uni-freiburg.de

Abstract. The inductive query language IQL is introduced. It is intended as a general, descriptive, declarative, extendable and implementable language for inductive querying that supports the mining of both local and global patterns, reasoning about inductive queries and query processing using logic, as well as the flexible incorporation of new primitives and solvers. IQL is an extension of the tuple relational calculus with functions, a typing system and various primitives for data mining. We hope that it will be useful as an overall specification language for integrating data mining systems and principles.

1 Introduction

The area of inductive databases [7], inductive query languages, and constraint-based mining [1] has promised a unifying theory and framework for reasoning about data mining principles and processes, which should also result in powerful inductive query languages for supporting complex data mining tasks and scenarios. The key idea is to treat patterns and models as first-class citizens that can be queried and manipulated. The slogan: "From the user point of view, there is no such thing as real discovery, just a matter of the expressive power of the available query language" has been advocated.

There has been a lot of progress in the past few years witnessed by the introduction of several inductive query languages, such as MINE RULE [10], MSQL [8], DMQL [6] and XMine [3]; Microsoft's Data Mining Extensions (DMX) of SQL Server [13]; the algebra of the 3W model [9]; the logic based query languages of MolFEA [11] and Datalog++ [5], which all have contributed new insights. MINE RULE, MSQL, DMQL and XMine focus on the derivation of either frequent itemsets or association rules; notation wise, these languages are extensions of the industry standard SQL language. Microsoft's SQL server includes a larger set of algorithms, and provides an interface for learning, clustering and applying a wider range of data mining algorithms, including association rules, decision trees and Bayesian networks. The algebra of the 3W model supports assocation rule discovery as well as learning rule based classifiers. The language of MolFEA allows for the discovery of patterns under constraints, and is more abstract. Datalog++ is somewhat similar to our proposal, but takes Datalog as its starting point and is less focused on the representation of constraints.

Despite this plethora of languages, there is still no comprehensive theory of inductive querying or a unifying query language. In this paper, we propose the inductive query language IQL, which addresses some of the limitations of existing approaches. We designed IQL with the following goals with in mind:

- to provoke discussion on inductive query languages;
- to encompass a rich variety of data mining tasks, including: local pattern mining over different domains, clustering, classification, regression as well as probabilistic modeling;
- to support reasoning about queries, their execution and their optimization using logic;
- to integrate data mining primitives in a database language; as database language we employ an extension of the tuple relational calculus rather than SQL because it is simpler and better grounded in theory;
- to design an extendable language, in which other researchers can describe and possibly implement their constraints, primitives and approaches; if this succeeds, IQL might become a unifying description language for data mining;
- to design an implementable language, even though we wish to stress that –at this point– we are not concerned with the efficiency of the resulting system but rather with the underlying abstract principles.

The paper is organized as follows: Section 2 provides an intuitive introduction to our query language; Section 3 introduces IQL in more detail. Within the IQL we believe that certain primitives should be supported. These are provided in Section 4. An intuition of how the query language could be evaluated is provided in Section 5; the kind of reasoning it supports is discussed in Section 6. A scenario is described in Section 7. Finally we conclude.

2 Some Example Queries

The best way to introduce the ingredients of a language, and hence also those of IQL, is by providing some examples. IQL is derived from and extends the query language we introduced earlier [11]. An example inspired on that language, but rewritten in IQL is :

create table R **as**

$$\{< \mathit{pattern} : S,\ \mathit{freq}1 : \mathit{freq}(S, D_1),\ \mathit{freq}2 : \mathit{freq}(S, D_2) > \ \mid\ S \in \mathit{Sequences} \wedge S \preceq \text{“}C - H - 0 - n\text{”} \ \wedge\ \mathit{freq}(S, D_1) \leq 0 \ \wedge\ \mathit{freq}(S, D_2) \geq 1\ \};$$

This query generates a relation in which the tuples consist of sequential patterns and their frequency in datasets D_1 and D_2. Furthermore, all patterns must occur at least once in dataset D_2, must not occur in D_1 and must be more general than (i.e., a substring of) “$C - H - O - n$” . This query thus corresponds to a typical local pattern mining step.

As a second example, consider

create view R' **as**

$$\{\, T + < target : D(T) > \mid D \in DecisionTrees[< A : Int, B : Int > \rightarrow < C : Int >] \wedge C45(D, R) \wedge T \in R \,\}.$$

This query creates a view R', which extends the relation R with the attribute *target*. The value of *target* is the prediction made by a decision tree generated by C45 on the projection of R to the attributes A, B and C (as the class attribute). So, this query does not only generate a decision tree but also applies it to a data set, which corresponds – in part – to a cross-over operation.

These two examples illustrate the following key ingredients of IQL:

- queries are generating relations of the form { tuple | condition (tuple) };
- IQL is an extension on the relational tuple calculus;
- the result of a query is a relation, hence, the closure property is satisfied;
- the values of the tuples can be complex, e.g. sequences, functions, etc.; we also allow for operations such as "+" and "-" on tuples, which join, respectively remove attributes from tuples;
- the traditional logical connectives such as $\wedge, \vee, \neg$ are permitted;
- IQL is able to employ functions; for instance,: $freq(t, D)$, which computes the frequency of the pattern t in the dataset D;
- classifiers (such as a decision tree) are regarded as functions; so IQL is not only able to employ functions, but also to generate and manipulate them;
- IQL employs a typing system; for instance the decision tree D maps tuples with attributes A and B onto their classes C;
- as in [4] there are virtual tables representing domains;
- as in the language by [11], there are some built-in predicates such as $\preceq$, which denotes generality, and $freq(p, D)$, which denotes the frequency of the pattern p in the dataset D;
- calls to specific algorithms, such as C45, can be integrated; this will be realized using templates, cf. Section 4.

Let us now define these ingredients in a more formal manner.

3 Manipulation of data

To manipulate data as well as pattern and functions, we shall employ an extension of the tuple relational calculus. The tuple relational calculus is a standard theoretical query language for relational databases. By using the relational calculus, we keep the desirable closure property: the result of each query is a relation. Furthermore, the relational calculus is based on logic and is therefore declarative.

Essential in the relational model is that data is stored in relations, each of which consists of a set of tuples. A tuple is an expression of the form $< n_1 : v_1, \ldots, n_k : v_k >$ where n_i is an attribute, and v_i a value out of the domain D_i

of the attribute n_i, e.g. the tuple $< a : 0,\ b : 1 >$. For reasons of convenience, we allow tuples to be joined or subtracted using the $+$ and $-$ symbols [1]. The schema of a tuple is denoted by $< n_1 : D_1, \ldots, n_k : D_k >$. For instance, in the above example, this is $< a : \mathit{Boolean}, b : \mathit{Boolean} >$. A relation is then a set of tuples over a particular schema, e.g. $R = \{\ < a : 0,\ b : 1 >, < a : 1,\ b : 0 >\ \}$. We will also say that the schema of a relation is $\{< n_1 : D_1, \ldots, n_k : D_k >\}$. In tuple relational calculus, variables range over tuples in relations.

The *syntax* of the tuple relational calculus is then defined as follows. A query is an expression of the form $\{t|q\}$, where t is a tuple and q is a formula. A *formula* is built from the traditional connectives $\wedge$, $\vee$ and $\neg$, and contains variables that can be quantified using the $\exists$ and $\forall$ quantifiers. The following atoms are allowed:

- atoms of the form $e_1 \theta e_2$, where $\theta \in \{\geq, \leq, >, <, =, \neq\}$ and e_i is a term. Constants, attributes a of tuples t (denoted by $t.a$) and tuples with one attribute can be used as terms.
- $t \in R$, where t is a tuple variable, and R is a relation.

From a data mining perspective, a dataset is conceived as a set of tuples, each of which contains information about an example. One crucial aspect of IQL is that we allow for arbitrary domains. For instance, we shall consider the domain of graphs, sequences, ..., and even those of particular functions. For such domains, there will typically be special built-in operators such as for instance the generality or covers relation $\preceq$ stressed by [11]. Similarly, we can conceive a pattern set as a set of tuples, each of which contains a pattern.

Even though we assume that the inductive database conceptually deals with domains such as graphs or sequences, this does not mean that we claim that an inductive database should be able to store such structures entirely in an attribute. For instance, an attribute in the graph domain could also be implemented as an identifier pointing to another relation storing the real graphs. At this point, we make an abstraction as to how such objects are incorporated or implemented, and essentially only assume that they can be manipulated and passed on using the IQL calculus.

From a data mining perspective, a crucial extension is that we allow for functions. Functions take tuples, relations and attributes as arguments. The *signature* of a function is therefore denoted by

$$(\mathit{type}_1, \ldots, \mathit{type}_n) \rightarrow \mathit{type}_{n+1},$$

where type_i either specifies a domain, or a schema. Predicates are functions where $\mathit{type}_{n+1} = \mathit{Boolean}$. Functions and predicates are incorporated in IQL by allowing expressions of the form $f(e_1, \ldots, e_n)$ where f is a function and the e_i are expressions with type type_i. These expressions can occur in atoms, as well as in tuples, as they denote particular values with type_{n+1}.

To deal with the special nature of classifiers, which, similar to patterns, can be conceived as domains, but can also be applied as functions, we introduce function

[1] For simplicity we shall assume that no clashes occurs (as e.g. in $< \mathit{temp} : 5 > + < \mathit{temp} : 6 >$).

domains. Similar to a pattern domain, a function domain has a name, for instance *DecisionTrees*, but also a signature, which defines the signature of an object that is used as function. An example is the function domain $DecisionTrees[< A : Int, B : Int > \rightarrow < C : Int >]$, which contains decision trees that can be applied to tuples with schema $< A : Int, B : Int >$.

The final elements of the IQL are:

- we order the domains in a hierarchy. For instance, the *DecisionTrees* and *SVMs* domains are both specializations of the *Classifiers* domain. We require that functions that operate on high-level abstract domains, also work on all specializations of that domain. This makes it possible to store several different types of classifiers in a single relation, and to apply all classifiers in this relation to classify the same examples.
- we allow 'overloading' of fuctions. This is convenient, for instance, as it allows to implement the *freq* and $\preceq$ functions for several pattern domains, without requiring new symbols.
- we introduce a *virtual relation* of schema $\{< element{:}D >\}$ for every data mining domain D (similar to [4]). These relations are necessary at the implementation level to make sure that the schema of every tuple variable in a query is known, and it is clear which overloaded function should be used;
- we allow new relations to be created that contain the result of a query, and we allow for the definition of views and functions.

The following example illustrates the last three points.

create function $f(id : Int)$ **as** $\{\ t - < id > \ | \ t \in D \wedge t.id = id\ \}$
create table F **as** $\{\ < pattern : S, id : v.id, freq : freq(S, f(v.id)) > \ | \ v \in ID \wedge S \in Sequences \wedge freq(S, f(v.id)) \geq 10\ \};$

In this example, the function f creates a subset of the database D having a particular value *id* for the corresponding attribute. The second query then looks for sequences that are frequent in the corresponding subsets of D. The *Sequences* relation is a virtual relation for the *Sequences* domain, which specifies the type of the tuple variable S in the query and clarifies that we are looking for frequent sequences. The type of S determines which definition of the overloaded *freq* function should be used.

4 Primitives

To introduce primitives that should be supported by the inductive database, we will use queries that we call *templates*. A template is a query that is solved by a data mining algorithm, if certain parameters are substituted. For example, a frequent itemset miner solves the following template problem:

$$\{\underline{I} | \underline{I} \in Itemsets \wedge freq(\underline{I}, \underline{R}) \geq \underline{\theta}\},$$

where the underlined terms, like $\underline{I}$, are terms that are allowed to be substituted. The underlying idea is that an algorithm can solve a certain query, if its template

matches the query; a template matches a query if the query can be obtained by substituting the terms in the template appropriately (i.e., such that types are taken into account). The above template introduces the domain *Itemsets*, and hence the types of functions should be specified. For this case, we could e.g. specify that *freq* is a function with the type

$$(Itemsets, \{< tid : Int, itemset : Itemsets >\}) \rightarrow Integer.$$

Overall, every data mining algorithm introduces models and patterns in a certain domain, introduces virtual relations over these domains, introduces functions and predicates that can be applied to the domain, and defines templates of queries that are solved by the data mining algorithm.

Local pattern mining In general, we can define that a frequent pattern mining algorithm implements the following meta-template:

$$\{\ \underline{P} \mid \underline{P} \in \underline{\underline{Pattern}} \wedge freq(\underline{P}, \underline{R}) \geq \underline{\theta}\ \};$$

this query itself is not implementable before $\underline{\underline{Pattern}}$ has been substituted with a derivative type that is a concrete domain, such as *Itemsets* or *Sequences*. Elements of a template that need to be substituted before the template represents an implementable algorithm, are double underlined.

Algorithms for mining under condensed representations, such as closed, free, or nonderivable itemset miners, implement the meta-template:

$$\{\underline{P} | \underline{P} \in \underline{\underline{Pattern}} \wedge freq(\underline{P}, \underline{R}) \geq \underline{\theta} \wedge \underline{\underline{representation}}(\underline{P}, \underline{\underline{Pattern}}, \underline{R})\};$$

here *representation* is a predicate with signature $(Pattern, \{< element : Pattern >\}, \{< tid : Integer, data : Pattern >\})$. Concrete miners for mining condensed representations substitute the $\underline{\underline{representation}}$ with the name of a representation, such as closed, free, generator, etc.

Observe that if a predicate *representation* is implemented, but not as part of a template, we can still find the condensed representation by postprocessing the output of a frequent pattern mining algorithm. Thus, the template determines whether the algorithm allows a constraint to be pushed in the search process or not.

The template that was solved by MolFEA is [11]:

$$\{\ \underline{S} \mid \underline{S} \in \underline{\underline{Pattern}} \wedge \underline{\phi}(\underline{S})\ \},$$

where ϕ is a boolean formula supported by MolFEA, which depends only on S and relations of the proper types. For every pattern domain, we believe that an inductive database should support the functions $\preceq$ and *freq*; furthermore, it should implement the frequent pattern mining template. This is sufficient to answer almost any local pattern mining query without pushing the constraint. For some pattern domains, for example, sequences, a solver can then be added to the database with a broader template, such as MolFEA, to allow the inductive database to push constraints in the mining process.

A recent branch of research involves that of mining top k patterns, where the top k patterns are determined according to some convex measure, such as the χ^2 test. A template which is answered by a top k free itemset miner is:

create view $\underline{R'}$ **as** $\{ < itemset : \underline{I},\ measure : \underline{measure}(\underline{I}, \underline{R}) > \mid \underline{I} \in Itemsets\ [< \underline{itemset} : Itemsets > \rightarrow < \underline{target} : Bool >] \wedge isFree(\underline{I}, Itemsets, \underline{R}) \};$

$\{\underline{I} | \underline{I} \in \underline{R'} \wedge rank(\underline{I}, \underline{R'}) \leq \underline{k}\}$

Here *rank* is a function that determines the rank of an itemset in a relation with attributes *itemset* and *measure* that is sorted according to this measure. For the *measure* function the 'target' attribute of an itemset has to be specified, with which a correlation is computed. This query illustrates that itemsets can also be interpreted as classifiers: they predict true iff the itemset is contained in an example; therefore, itemsets can be specialized as classifiers. The relation R is supposed to contain a *target* attribute. The *isFree* function, as previously defined, assumes that R is a relation containing itemsets. Thus, it can also be applied on a relation with an additional *target* attribute.

Observe that the view will not be materialized if we have an algorithm that matches this template. However, assume that the user adds a minimum frequency constraint in the view, then an alternative query evaluation plan may arise, in which the view is evaluated first, and then postprocessed.

Classification algorithms In general, an algorithm for learning a classifier implements the following template:

$\{ \underline{C} \mid \underline{C} \in \underline{\underline{Classifiers}}\ [< \underline{X} > \rightarrow < \underline{target} : String >] \wedge \underline{\underline{AlgorithmName}}(\underline{C}, \underline{R}) \};$

here *Classifiers* is the name of the classification model, for example, *DecisionTrees*; *AlgorithmName* refers to the algorithm that learns the decision tree, for example $C45$. Relation R contains, or refers to, the examples from which the classifier is to be learned. This relation must have a scheme compatible with that of the classifier that is learned (i.e., the set of attributes in the relation must be a superset, and of corresponding types).

To deal with additional constraints on the model that is learned, additional functions can be introduced. A typical function that should be supported by an inductive database is the accuracy function:

$$accurary(C, D) = |\{T | T \in D \wedge C(T) \neq T.\text{“}target(C)\text{”}\}|/|D|,$$

which counts the number of examples in dataset D for which the target predicted by classifier C does not match the actual value of this attribute; “$target(C)$” represents the name of the attribute in the righthand side of the signature of the classifier.

For particular classifiers, such as decision trees, additional functions should be supported. In the case of decision trees, the $size(T)$ should return the size of the decision tree.

A constraint on the accuracy of a classifier can now easily be applied through atoms like $accuracy(C, D) \leq 0.9$. However, for many algorithms, such as decision tree learners, this will mainly mean that their result will be filtered; if the constraint is not satisfied, the decision tree is removed. Some algorithms, however, may choose to take into account constraints on accuracy and size [12]. The templates of these algorithms are specializations of the template above, also matching atoms involving the *accuracy* and *size* functions.

Probabilistic Models Probabilistic models differ from classifiers in that they do not output a single class, but a probability distribution over some target attributes.

$$\{\ \underline{C} \mid \underline{C} \in \underline{\underline{ProbModels}}\ [< \underline{X} > \rightarrow \{< \underline{target} : String, probability : Float >\}\ ] \wedge \underline{\underline{AlgorithmName}}(\underline{C}, \underline{R})\ \};$$

This template reflects this property; a probabilistic model returns a relation, in which for each target attribute class a probability is stored. An inductive database should support facilities for using such probabilistic models as classifiers though we shall not go into the syntactic details of this here.

Clustering Clusterings are similar to probablistic models in that they do not target a specific class attribute, but rather try to find meaningful groups within the data. For instance, assume that we have a k-means clustering algorithm that puts examples into multiple clusters and assigns a degree of membership for each cluster (for example, according to the distance to the cluster centre). Then the following template formalizes such an algorithm:

$$\{\underline{T} + \underline{L} \mid \underline{C} \in KMeansClusterings[\ \underline{X} \rightarrow cluster : Int, membership : Float\ ] \wedge myKMeans(\underline{n}, \underline{C}, \underline{R}) \wedge \underline{T} \in \underline{R} \wedge \underline{L} \in \underline{C}(\underline{T})\},$$

where X is a subset of the attributes of R and n is the number of clusters in the k-means clustering. For each example this query outputs the clusters that it is in, and the degree of this membership. Observe that in this template, learning and prediction are combined into one template. For many clustering algorithms, it is difficult to separate these operations. However, if a clustering algorithm generates a function that can assign clusters to unseen examples, then it can be handled as a classifier.

5 Query Evaluation

Until now we have mainly focused on the elements of IQL. In this section, we want to argue that the language is also executable and implementable. In this preliminary draft, we focus on the following subset of IQL:

- all tuple variables are either existentially quantifier or do occur in the left-hand side of the query;

– if the query is written in disjunctive normal form, every conjunction must contain a non-negated atom $t \in R$ for every tuple variable occuring in the conjunction.

A central concept in database theory is that of query safety, which states that the result of each query should be finite. Even though we shall not provide a formal definition of safety for IQL queries, under the restrictions specified above, IQL queries are safe provided that all relations are finite. This will clearly not be the case when dealing with virtual relations.

An essential observation is that templates can be seen as definitions of functions. Every variable that occurs in the lefthand side of the template, can be considered an output of the function; all other variables and relations are inputs. Assume that a template can be *matched* against the data, i.e., there is a substitution such that from the template a subset of the atoms of the query can be obtained, then we can replace this part of the query with a call to a function that represents the data mining algorithm. As long as data mining algorithms then generare finite results, the query language is safe.

Overall, we can now proceed as follows. Without loss of generality we assume that the formula is in disjunctive normal form. The result of the query then consists of the union of the results of the individual conjunctions. We call the tuple variables that occur in the lefthand side of the original query the *projection variables*. To make the union well-defined, we require that the projection variables occur with the same schema in all conjunctions.

The query is executable if we can match the templates in a non-overlapping way, such that all references to virtual data mining relations are matched; then, each set of atoms that is matched, is replaced by an appropriate function call to the data mining algorithm; a tuple variable that was part of the output of the template, is replaced by a tuple variable that ranges over the result of the data mining algorithm[2]; We can evaluate the query by walking recursively from left to right through the query, constantly maintaining an assignment for all tuple variables. If we encounter an atom $T \in R$, we recurse on every possible assignment of T. After we have evaluated the entire query for an assignment to all tuple variables, the projection of that assignment can be computed and stored as a result of the query.

6 Reasoning

Due to its embedding in logic, IQL allows one to reason about queries, as [11]. For instance, consider the sequence :

$$\textbf{create table } R \textbf{ as } \{ \ < pattern : S, freq : freq(S, D_1 \cup D_2) > \ | \ S \in Sequences \wedge freq(S, D_1 \cup D_2) \geq 5 \ \};$$

$$\{< pattern : S, freq : freq(S, D_1) > | S \in Sequences \wedge freq(S, D_1) \geq 5\};$$

[2] We still assume that name clashes are absent.

and assume that the queries are posed sequentially. Then one can actually see that the answer to the first query is a superset of that of the second one. Therefore, rather than calling the frequent pattern miner again for the second query, one might simply go through the result of the first one to verify which patterns satisfy the second ferquency constraint. Examples of this kind of reasoning, and a deeper discussion of these issues, is provided in [11]. Observe, however, that the frequencies of all frequent sequences have to be computed to finally answer the second query, as the frequencies in the second query may be smaller than in the first.

In IQL, this type of reasoning can be extended to constraints on other domains. For instance, a decision tree with minimum accuracy 0.9 on a dataset R is also a decision tree with minimum accuracy 0.8 on the same dataset.

Due to its close connection to relational calculus, there are similar optimization possibilities in IQL as in relational calculus. For instance, consider this query:

$$\{T + C(T) | T \in R \wedge C \in DecisionTrees[< A : Int > \rightarrow < B : Int >] \wedge C45(C, R)\},$$

to evaluate this query, the query optimizer should first construct the decision tree, and then apply it to all examples; it should not choose to construct the decision tree repeatedly for every example again.

Furthermore, the query optimizer should be aware that it is desirable to push a constraint in an algorithm if possible; for example, if a specialized closed itemset miner is available, it should be used. At this point, it remains an open question and important topic for further research as to what good optimization strategies for query evaluation are.

7 Scenario

IQL should support the description of scenarios [2]. In this section we will demonstrate a typical scenario, in which a pattern miner is used to find frequent patterns, these frequent patterns are then used to create features, and finally a classification model is learned.

The first step in this scenario is easily described. Assume that we have a database of molecules *HIV*, and we are looking for subgraphs with a high support in *active* molecules, but a low support in the *inactive* molecules:

create function $hiv(d : String)$ **as**
$\{ T- < activity > \mid T \in HIV \wedge T.activity = d \}$

create table R **as**
$\{ S \mid S \in Graphs[< graph : Graphs > \rightarrow < target : Boolean >] \wedge$
$freq(S, hiv(active)) \geq 10 \wedge freq(S, hiv(inactive)) \leq 10 \}$

The subgraphs S are conceived to be classifiers of relations containing a *graphs* attribute; they predict 1 iff the *graphs* attribute contains the specified subgraph.

For the second step of the scenario we require an additional construction in the language. To apply a classification algorithm, we need to create a table in which every column corresponds to one pattern in the relation R. For this we need a construction for creating tables with arbitrary dimension. To deal with this issue, we allow tuples of the form $< value_1 : value_2 >$ in the lefthand side of queries, in which an attribute is created that obtains its name $value_1$ from an attribute of another table and its value as usual. This allows us to formulate the following query:

create function $f(Data : \{< id : Int, graph : Graphs, activity : String >\})$**as**
$\{< id : T.id, string(G) : G(T), activity : T.activity > \mid T \in Data \wedge G \in R\}$

create table $Features$ **as** $f(HIV)$

In this query, we assume that the *string* function creates an appropriate name for the subgraph. This query can be evaluated in two steps. First, a relation with schema $\{< id : T.id, name : String, value : Bool, activity : T.activity >\}$ can be created. From the *name* attribute in this table, the names of the new attributes are collected. Then, for every grouping of tuples according to *id* and *activity*, all $< name, value >$ values are collected, and used to fill in the row.

The result of this query is a relation in which columns denote whether a graph contains a certain subgraph or not. We can build a decision tree for this relation.

create table R' **as**
$\{D | D \in DecisionTrees$
$[schema\text{-}of(Features) - < id, activity > \rightarrow < activity : Bool >] \wedge C45(D, Features)\}$.

Here, *schema-of* returns the schema of relation *Features*; the classifier should use the features in this relation, excluding the *id* and *activity* attributes. Finally, we can use this decision tree to predict the activity of molecules in a dataset HIV'.

create table *HIVPredictions* **as**
$\{T' + D(T) | T \in f(HIV') \wedge D \in R' \wedge T' \in HIV' \wedge T.id = T'.id\}$.

This query shows how using traditional data manipulation operations, we can associate the prediction of a molecule to its original representation, instead of to its binary feature representation.

8 Conclusions

We presented a relational calculus for data mining. The key ingredients were the inclusion of functions, and the use of models as functions in the relational tuple calculus. This allowed us to integrate a large set of algorithms into IQL, including classification algorithms and clustering algorithms.

Even though IQL was presented in a rather informal way, we believe that IQL can already be used as a description language and interface to a wide variety of

data mining algorithms and techniques in a uniform and theoretically appealing way. The authors would also like to herewith invite other groups interested in the development of inductive query languages to describe their favorite constraint based mining tools within IQL.

Nevertheless, there are many remaining issues for further research concerning IQL. One of these concerns a more formal definition of the syntax and semantics of IQL, which we could not work out here due to space restrictions. Another concerns the extension of the evaluation strategy to more general IQL statements, and the development of an optimization strategy grounded in logical reasoning. Finally, an implementation of IQL could help to support our claims even though implementation and efficiency are not the primary aim of this work.

Acknowledgements This work was supported by the EU FET IST project IQ ("Inductive Querying"), contract number FP6-516169.

References

1. F. Bonchi and J-F. Boulicaut. *Knowledge Discovery in Inductive Databases, 4th International Workshop, Revised Selected and Invited Papers.* LNCS 3933. 2006.
2. J-F. Boulicaut, L. De Raedt, and H. Mannila. *Constraint-Based Mining and Inductive Databases, European Workshop on Inductive Databases and Constraint Based Mining.* LNCS 3848. 2004.
3. D. Braga, A. Campi, A. Ceri, S. Lanzi, and M. Klemetinen. Mining association rules from XML data. In *Proceedings of the 4th International Conference on Data Warehousing and Knowledge discovery*, LNCS 2454, 2002.
4. T. Calders, B. Goethals, and A. Prado. Integrating pattern mining in relational databases. In *PKDD*, 2006.
5. F. Giannotti, G. Manco, and F. Turini. Specifying mining algorithms with iterative user-defined aggregates. In *IEEE Transactions Knowledge and Data Engineering*, pages 1232–1246, 2004.
6. J. Han, Y. Fu, K. Koperski, W. Wang, and O. Zaiane. DMQL: A data mining query language for relational databases. In *Proceedings of the ACM SIGMOD Workshop on research issues on data mining and knowledge discovery*, 1996.
7. T. Imielinski and H. Mannila. A database perspective on knowledge discovery. In *Communications of the ACM*, volume 39(11), pages 58–64, 1996.
8. T. Imielinski and A. Virmani. MSQL: A query language for database mining. In *Data Mining and Knowledge Discovery*, volume 2(4), pages 373–408, 1999.
9. T. Johnson, L. V. Lakshmanan, and R. Ng. The 3w model and algebra for unified data mining. In *Proc. VLDB Int. Conf. Very Large Data Bases*, pages 21–32, 2000.
10. R. Meo, G. Psaila, and S. Ceri. An extension to SQL for mining association rules. In *Data Mining and Knowledge Discovery*, volume 2(2), pages 195–224, 1998.
11. L. De Raedt. A perspective on inductive databases. In *SIGKDD Explorations*, volume 4(2), pages 69–77, 2003.
12. J. Struyf and S. Dzeroski. Constraint based induction of multi-objective regression trees. In *KDID*, pages 222–233, 2005.
13. Z. Tang and J. MacLennan. *Data Mining with SQL Server 2005.* Wiley, 2005.

Mining Correct Properties in Incomplete Databases

François Rioult and Bruno Crémilleux

GREYC, CNRS - UMR 6072, Université de Caen
F-14032 Caen Cédex France
{Francois.Rioult,Bruno.Cremilleux}@info.unicaen.fr

Abstract. Missing values issue in databases is an important problem because missing values bias the information provided by the usual data mining methods. In this paper, we are searching for mining patterns satisfying correct properties in presence of missing values (it means that these patterns must satisfy the properties in the corresponding complete database). We focus on k-free patterns. Thanks to a new definition of this property suitable for incomplete data and compatible with the usual one, we certify that the extracted k-free patterns in an incomplete database also satisfy this property in the corresponding complete database. Moreover, this approach enables to provide an anti-monotone criterion with respect to the pattern inclusion and thus design an efficient level-wise algorithm which extracts correct k-free patterns in presence of missing values.

1 Introduction

Missing values in databases is a problem as old as the origin of these storage structures. It is an important issue because information extracted by usual data mining or statistics methods in incomplete data are biased and do not reflect the sound knowledge on the domain. We show in Section 2.2 the damages due to missing values in the pattern mining area. The popular uses of (frequent) patterns (e.g., rules, classification, clustering) are no longer reliable. The basic idea of elementary techniques to cope with missing values is to guess them (e.g., use of the mean, the most common value, default value) and complete them. Unfortunately, these techniques are not satisfactory because they exaggerate correlations [1] and missing values completion remains a hard track.

On the contrary, in this paper, we are searching for mining patterns satisfying properties in presence of missing values which are also satisfied in the corresponding complete database. Our key idea is to highlight properties from an incomplete database, these properties must be consistent in the real database without missing values. We say that these properties are *correct*. This can be achieved because some characteristics are not removed by missing values. For instance, if a pattern is frequent in a database with missing values, it must be frequent in the corresponding complete database. In Section 3.1, we propose an operator to define the relation between an incomplete database and every possible completion.

In this paper, we focus on the property of k-freeness [2]. This property is on the core of frequent pattern mining, association rules building, and more generally *condensed representations* of frequent patterns [3] which enable multiple uses of frequent patterns. Our main contribution is to propose a new definition of the k-freeness property in incomplete data which is fully compatible with the usual one in a database without missing values.

This new definition certifies that the extracted patterns satisfying this definition in an incomplete database are k-free in the corresponding complete database and, in fact, in every completion of the incomplete database. Moreover, this approach leads to an anti-monotone criterion with respect to the pattern inclusion and thus allows to design an efficient level-wise algorithm which extracts k-free patterns in presence of missing values. This work is a first step toward classification in incomplete databases with generalized associations and its application to missing values imputation.

The presentation is organized as follows: Section 2 gives the background about the k-freeness of patterns, briefly shows the damages caused by the missing values and presents our position statement. Section 3 defines the computation of k-free patterns in presence of missing values and demonstrates that these patterns are correct in the corresponding complete database. Experiments on benchmark data confirm the effectiveness of our method (Section 4).

2 Preliminaries

In this section, we introduce the k-free patterns and the generalized association rules which stem from these patterns. We show the damages due to the missing values and we give our position statement to solve this pattern mining problem.

Let us consider a database which gathers *objects* depicted by quantitative or qualitative *attributes* in an *attribute/values* format (see Table 1). Eight objects are described by three attributes X_1, X_2 and X_3. In the field of boolean pattern mining, qualitative attributes need to be discretized in order to get boolean contexts (this article does not discuss this stage).

	attributes		
objects	X_1	X_2	X_3
o_1	$+$	$\rightarrow$	0.2
o_2	$-$	$\rightarrow$	0
o_3	$+$	$\rightarrow$	0.1
o_4	$+$	$\leftarrow$	0.4
o_5	$-$	$\rightarrow$	0.6
o_6	$-$	$\rightarrow$	0.5
o_7	$+$	$\leftarrow$	1
o_8	$-$	$\leftarrow$	0.8

Table 1. Attribute/value format database.

	attributes						
objects	a_1	a_2	a_3	a_4	a_5	a_6	a_7
o_1	×		×		×		
o_2		×	×		×		
o_3	×		×		×		
o_4	×			×		×	
o_5		×	×			×	
o_6		×	×			×	
o_7	×			×			×
o_8		×		×			×

Table 2. Boolean context r.

Let r be a database and $(\mathcal{A}, \mathcal{O}, R)$ a *boolean context* where $\mathcal{O}$ is the set of objects, $\mathcal{A}$ is the set of attributes and R is a binary relation. An object is a subset of $\mathcal{A}$ (for example, $o_1 = \{a_1, a_3, a_5\}$) and it will be denoted as a string (i.e., $a_1a_3a_5$). $|r|$ is the number of objects in r, *i.e.* $|r| = |\mathcal{O}|$. Table 2 indicates the boolean context where X_3 is coded by the attributes a_5 to a_7.

A *pattern* X is a subset of $\mathcal{A}$, its *support* is the set of objects containing X (we denote $supp(X) = r_X = \{o \in \mathcal{O} \mid X \subseteq o\}$) and its *frequency* $\mathcal{F}(X) = |supp(X)|$ is the number of objects in the support. A *classical* association rule [4] is an expression $X \rightarrow Y$, where X and Y are two patterns. It is quantified by its frequency (i.e., $\mathcal{F}(X \cup Y)$) and its confidence: $conf(X \rightarrow Y) = \mathcal{F}(X \cup Y)/\mathcal{F}(X)$.

2.1 Generalized association rules and k-freeness

We start by recalling generalized patterns [2] because they are at the core of the generalized association rules. A *generalized pattern* is made of boolean attributes and negations of boolean attributes. For example, the generalized pattern $Z = a_1\overline{a_2}a_3$ can be written as the union of a *positive* part $X = a_1a_3$ and a *negative* one $\overline{Y}$ where $Y = a_2$. An object o supports $Z = X \cup \overline{Y}$ if $X \subseteq o$ and $Y \cap o = \emptyset$. To alleviate the notations, we omit the union sign in the following and write $X\overline{Y}$ instead of $X \cup \overline{Y}$. $\mathcal{F}(X\overline{Y})$ is central: if it is null, one element of Y is always present with X and ensures a generalized association between X et Y. These associations lead to the *generalized association rules* introduced in [2] which are a generalized form of association rules. The originality of these rules (also called *disjunctive rules*) is to conclude on a disjunction of attributes as indicated by Definition 1, which comes from [2].

Definition 1. *A generalized association rule based on $Z = X \cup Y$ is an expression $X \rightarrow \vee Y$ where X and Y are two classical patterns. It is exact in a database r if every object of r containing the premise X also contains one attribute of the conclusion Y. We denote $\models_r X \rightarrow \vee Y \iff \mathcal{F}(X\overline{Y}, r) = 0$.*

We define the frequency of a generalized association rule as follows (this definition diverges with that of the classical association rules).

Definition 2. *The frequency $\mathcal{F}(X \rightarrow \vee Y)$ of $X \rightarrow \vee Y$ is the number of objects containing X and at least one attribute of Y. We get $\mathcal{F}(X \rightarrow \vee Y) = \mathcal{F}(X) - \mathcal{F}(X\overline{Y})$.*

Let us move now to k-free patterns. They have been proposed[1] by Calders and Goethals [2], and they are very useful to compute the generalized association rules. A k-free pattern expresses the absence of correlation between its attributes:

Definition 3 (k-free pattern). *A pattern Z is k-free in a complete database r (without missing values) and we denote $kFree(Z, r)$ if it does not exist any generalized association rules based on Z in r, or: $\forall X \cup Y = Z,\ |Y| \leq k \Rightarrow \mathcal{F}(X\overline{Y}) \neq 0$.*

The k-free patterns have excellent properties to sum up the collections of frequent patterns. For example, in the `mushroom` dataset [6], there are $2.7 \cdot 10^9$ present patterns, but $426,134$ 1-free and $224,154$ 2-free patterns. With k higher than 5, the number of k-free patterns stays at $214,530$, and they are mined in two minutes. Until now, k-free patterns have mostly been employed to compute condensed representations of frequent patterns [3] but they get meaningful properties to produce rules. In particular, 1-free patterns are used to compute the non redundant classical association rules [7,8]. The premise of such a rule is a 1-free X and its conclusion is the GALOIS closure $h(X)$. The exhibition of non redundant generalized association rules is more complex. We indicate two techniques. The first mines the 1-free patterns and then compute their *generalized closure* [9]. It gathers all minimal patterns Y sharing one attribute with every object containing X, it is obtained by computing the minimal transversals [10] of these objects [9]. The second technique takes benefit from the anti-monotonicity of the k-freeness and the border theory of this property [11]. The rules are built from the minimal non k-free patterns, which

[1] With $k = 2$, these patterns have been introduced by [5] with the term of *disjunction-free sets*.

constitute the *negative border of the k-free patterns* (details are given in [12]). Generalized association rules stem from non k-free patterns (such a rule $X \rightarrow \vee Z \backslash X$ is built from a non k-free pattern Z where X is the smallest subset of Z such that $\mathcal{F}(X\overline{Z \backslash X}) = 0$).

Generalized association rules convey correlations with a richer formalism than the classical ones. They enable new uses such as supervised classification [13] based on positive and negative rules [14] (i.e., rules concluding on an attribute or its negation). For example, the rule $a_1 \rightarrow a_4 \vee a_5$ is exact in the data of Table 2 and leads to the positive rule $a_1\overline{a_4} \rightarrow a_5$ and the negative one $\overline{a_4 a_5} \rightarrow \overline{a_1}$.

From the computation point of view, k-freeness is an anti-monotone property and these patterns can be efficiently mined thanks to the levelwise framework [11]. In order to check if a candidate pattern is k-free during the scan stage, the frequency of $X\overline{Y}$ is computed with the inclusion-exclusion principle [15], by using the frequencies of the subsets of XY: $\mathcal{F}(X\overline{Y}) = \sum_{\emptyset \subseteq J \subseteq Y} (-1)^{|J|} \mathcal{F}(XJ)$. As we have seen that in practice k remains low, the difficulty of computing the supports with the inclusion-exclusion principle is endurable.

2.2 Damages of missing values on k-free patterns

We show now the damages due to the missing values. Assume that some attributes of the dataset given in Table 1 are unknown, then missing values appear. We use the character `'?'` to denote that a value is neither present nor absent for *every* boolean attribute coming from the corresponding attribute in the original database. We have introduced three missing values in our running example and the database r' resulting from this operation is indicated in Table 3.

	attributes						
objects	a_1	a_2	a_3	a_4	a_5	a_6	a_7
o_1	×		×		×		
o_2		×	×		×		
o_3	×		×		**?**	**?**	**?**
o_4	×			×		×	
o_5		×	×			×	
o_6	**?**	**?**	×			×	
o_7	×			×			×
o_8		×	**?**	**?**			×

Table 3. Incomplete DB r'.

Complete DB r				Incomplete DB r'			
X	$h(X)$	X	$h(X)$	X	$h(X)$	X	$h(X)$
a_1		a_1a_3	a_5	a_1		a_1a_3	
a_2		a_1a_4		a_2		a_1a_5	
a_3		a_1a_5	a_3	a_3		a_2a_3	
a_4		a_2a_3		a_4	a_1	a_2a_5	a_3
a_5	a_3	a_2a_6	a_3	a_5	a_3	a_2a_6	a_3
a_6		a_3a_6	a_2	a_6		a_3a_6	
a_7	a_4			a_7		a_4a_7	

Table 4. 1-free 2-frequent patterns and their closures.

The usual support computation for a pattern X in an incomplete database is realized as follows: an object belongs to the support of X if all of its attributes are present in X. If one of its attributes is missing or absent, the object does not belong to the support. How to compute in presence of missing values the supports for generalized patterns? Definition 3 does not plan this situation and the problem is particularly accurate for computing the frequency of $X\overline{Y}$. Without any recommendation, computations are performed by ignoring the missing values (i.e, they are not taken into account).

Table 4 depicts this problem. This table gives the 1-free patterns with a minimum support of two objects. The left part relates the results in the complete database, the right

part in the incomplete one. For each pattern, its closure is indicated. The right part lists the 1-free patterns of r': a_1a_4 is 1-free in r and no more in r'. Furthermore, the right part includes patterns, such as a_2a_5 and a_4a_7, which are not in r: we qualify them as *incorrect*.

Missing values lead to damages both on free patterns and their closures. Assume that an attribute a belongs to X's closure in the complete database: it means that a is always present with X. If missing values appear on a, it may happen that this association is broken for some objects: a goes out from the closure (damage on the closure) and Xa can become free (damage on the free pattern). In our example, a_4 is in a_7's closure in r, while it goes out from this closure in r' because of the missing value in the object o_8. Thus, a_4a_7 is incorrectly declared 1-free.

Experiments on benchmarks from the UCI [6] emphasize these damages as well. Starting from a complete database, we artificially introduce missing values according to a uniform probability. Then we mine the 3-free patterns and measure the number of incorrect patterns relatively to the number of correct patterns in the original context (cf. Figure 1). The number of incorrect patterns differs according to the databases. It is less than 10% for the datasets `pima`, `wine`, `liver-disorders`, `servo` and `tic-tac-toe` (the corresponding chart is not reported). For the datasets given on the left part of the figure, the number of incorrect patterns is between 10 and 90% of the number of exact patterns. In the right part, this quantity goes to 300%, what means that for four computed patterns, three are incorrect.

In real conditions where the complete database is not known, it is impossible to differentiate good and bad patterns, and to say in advance if a small or a big proportion of incorrect patterns will appear. Our work aims at avoiding the damages by correctly computing the k-free patterns in incomplete contexts.

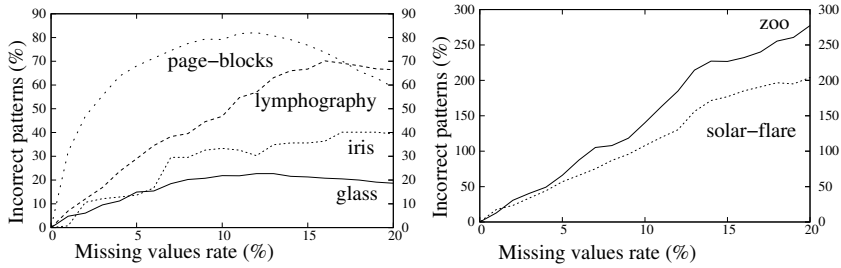


Fig. 1. Incorrect 3-free patterns in UCI datasets.

2.3 Position of our work

There are several works which address the missing values issue in databases [16,17] but contributions in the field of data mining are few. Arnaud RAGEL [18] studied association rules mining in presence of missing values by redefining the support and the confidence, these rules may be used to a *completion* (or *imputation*) goal. More recently, [19] gives a basic completion method, founded on the probability of the different attributes. The

support of a pattern for an object is no more boolean but probabilistic. [20] computes prediction rules in the complete part of a database. These rules provide intervals for continuous attributes.

Our work stems from the following principles:

- we do not want to impute the missing values before the knowledge discovery stage, because it is a difficult operation without any specific knowledge.
- we wish to mine the whole incomplete database without reducing it to its complete part. It means that we do not want to remove objects or attributes.

We do not assume any statistical hypothesis about the probability model of the missing values. In order to deal with missing values, the next section defines a *modeling operator* $mv()$. We will see that this formalization is useful because it allows to define an incomplete database as the result of an operation removing some values from the complete database. Then computations performed in an incomplete database can characterize properties which are common to every corresponding complete database.

The following shows that it is possible to discover valid knowledge for the complete database under these hypothesizes. As stated in introduction, this principle is not surprising: if we consider that missing values hide the true values of the data, the frequencies of some patterns will only decrease (we do not know for some objects if they are present). A frequent pattern in an incomplete database only can be *a fortiori* frequent in the complete dataset. We will use the same principle to compute correct k-free patterns in presence of missing values.

3 Mining k-free patterns in incomplete databases

We propose here a definition of the k-freeness property in an incomplete database. We show that it enables to compute patterns ensuring the property of freeness in every completion.

3.1 Missing values modeling operator

As previously explained, our position for the missing value problem requires a modeling operator. It defines the relation between an incomplete database and every possible completion.

Definition 4 (Missing values modeling operator). *Let $r = (\mathcal{A}, \mathcal{O}, R)$ be a boolean context. An operator $mv()$ is named a missing values modeling operator if it transforms a complete database r in $mv(r) = (\mathcal{A}, \mathcal{O}, mv(R))$. The new binary relation $mv(R)$ takes its values in $\{present, absent, missing\}$ and satisfies the following properties, for every attribute a in $\mathcal{A}$, every object o in $\mathcal{O}$, and $value \in \{present, absent\}$:*

1. $mv(R)(a, o) = value \Rightarrow R(a, o) = value$;
2. $R(a, o) = value \Rightarrow mv(R)(a, o) \in \{value, absent\}$;

Section 2.2 showed that computing the k-free patterns without precaution leads to incorrect patterns. In our work, we *correctly* define the computation of the k-freeness property:

Definition 5 (k-correct pattern). *Let r' be an incomplete database and $mv()$ a modeling operator for the missing values. A pattern Z is k-correct in r' if for every complete database r, $(mv(r) = r') \Rightarrow kFree(Z, r)$.*

3.2 Temporarily deactivating objects

We introduce here the *deactivation* of objects in an incomplete database. It differentiates on the one hand the objects which support or not a given pattern, and on the other hand the incomplete objects where the decision of support can not be taken. The deactivation enables to quantify the frequency gap between the complete and the incomplete database. In presence of missing values, the frequencies can indeed only decrease. In our example (Table 2), $\mathcal{F}(a_3a_5, r) = 3$ but $\mathcal{F}(a_3a_5, mv(r)) = 2$ (Table 3 with $r' = mv(r)$). In order to correctly compute the frequency of a pattern X in $mv(r)$, it is necessary to differentiate the objects of $mv(r)$ having a missing value among the attributes of X. These objects will be temporarily deactivated in order to compute an estimation of $supp(X, r)$ with the help of $supp(X, mv(r))$, because it is impossible to decide if they do contain X or not.

Definition 6 (Deactivated object). *For a classical pattern $X \subseteq \mathcal{A}$, an object $o \in \mathcal{O}$ is deactivated if $\forall a \in X, mv(R)(a, o) \neq absent$ and $\exists a \in X$ s.t. $mv(R)(a, o) = missing$. We denote $\mathcal{DES}(X, mv(r))$ for the objects of $mv(r)$ deactivated for X.*

Figure 2 exemplifies the notion of deactivation, by simultaneously presenting the complete database r (on the left) and the incomplete one $mv(r)$ (on the right). We suppose that each object of the top part contains X and this part is named r_X. The down part is named $r_{\overline{X}}$.

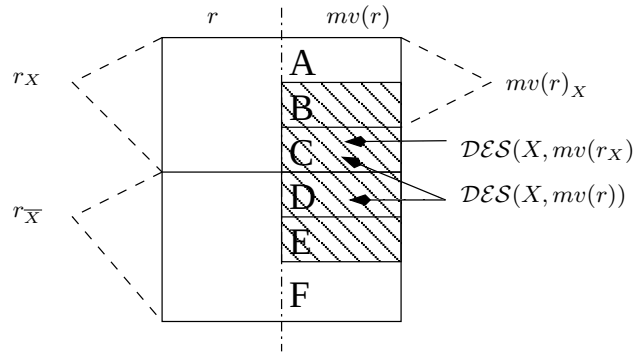


Fig. 2. Database $mv(r)$ and deactivated objects for X.

On the right, the hatched zone shows the objects of $mv(r)$ which contain missing values. It is composed of six sets of objects, which are described below (their composition is indicated for our example of Table 3, with $X = a_2a_3$):

Region A: (o_2, o_5) the objects without missing value, containing X;

Region B: (no object in our example) the objects initially containing X, whose missing values do not obscure the presence of X. These objects belong to $mv(r)_X$;
Region C: (o_6) the objects initially containing X, whose missing values hide the presence of X and constitute $\mathcal{DES}(X, mv(r_X))$;
Region D: (o_8) the objects not containing X in the complete database, but which could contain it with a suitable imputation of the missing values. The object o_8 does not contain the pattern a_2a_3 in the complete database of our example, and it is preventively deactivated;
Region E: (o_3) the incomplete objects not containing X in the original dataset nor after any imputation of the missing values;
Region F: (o_1, o_4, o_7) the complete objects which do not contain X.

In the incomplete database $mv(r)$, each object is assigned in three different groups for deciding the support of X:

Regions A and B: $mv(r)_X$ the objects supporting X, in spite of the missing values of B;
Regions C and D: $\mathcal{DES}(X, mv(r))$ the objects where the support of X is undecidable;
Regions E and F: the objects not supporting X.

The deactivation allows to precisely characterize the support difference between the incomplete database and the complete one:

Proposition 1. *Let X be a classical pattern, r a database and mv a modeling operator. $\mathcal{DES}(X, mv(r_X)) = r_X \setminus mv(r)_X$ and $|\mathcal{DES}(X, mv(r_X))| = \mathcal{F}(X, r) - \mathcal{F}(X, mv(r))$.*

Let us detail this principle for our example and the pattern a_2a_3: $r_{a_2a_3} = \{o_2, o_5, o_6\}$ and its frequency is 3. In the incomplete database, its frequency is 2 and $\mathcal{DES}(a_2a_3, mv(r_{a_2a_3})) = \{o_6\}$: we have the equality of Proposition 1. If the complete dataset r is not known, r_X is neither known, nor $|\mathcal{DES}(X, mv(r_X))|$. But the support can be bounded with considering the deactivated objects in $mv(r)$ instead of $mv(r_X)$, because this database contains more objects than $mv(r_X)$. In our example $\mathcal{DES}(a_2a_3, mv(r)) = \{o_6, o_8\}$ because of the confusion induced in o_8 by the missing value on a_3 and a_4. $\mathcal{F}(a_2a_3, r)$ is then between $\mathcal{F}(a_2a_3, mv(r))$ and $\mathcal{F}(a_2a_3, mv(r)) + |\mathcal{DES}(a_2a_3, mv(r))|$, i.e. between 2 and 4.

In the following, it is necessary to define the deactivation for the generalized patterns. For that purpose, we use the inclusion-exclusion principle:

Definition 7 (Generalized deactivation).
$des(X\overline{Y}, mv(r_{X\overline{Y}})) = \sum_{\emptyset \subseteq J \subseteq Y} (-1)^{|J|} |\mathcal{DES}(XJ, mv(r_{XJ}))|$.

The set $\mathcal{DES}(X\overline{Y}, mv(r_{X\overline{Y}}))$ is not defined, so we denote the generalized deactivation with lower cases: $des(X\overline{Y}, mv(r_{X\overline{Y}}))$. It allows nevertheless to quantify the frequency difference between the complete and the incomplete database.

Proposition 2. $des(X\overline{Y}, mv(r_{X\overline{Y}})) = \mathcal{F}(X\overline{Y}, r) - \mathcal{F}(X\overline{Y}, mv(r))$.

This frequency gap can be negative. When the association between X and Y exists in the complete database ($\mathcal{F}(X\overline{Y}, r) = 0$), one missing value can delete it in the incomplete one ($\mathcal{F}(X\overline{Y}, mv(r)) > 0$). In this case, the difference is negative. In our example, $des(a_7\overline{a_4}) = 0 - 1 = -1$.

For the deactivated objects regarding an association $X \rightarrow \vee Y$, we define $|\mathcal{DES}(X \rightarrow \vee Y, mv(r_{X \rightarrow \vee Y}))| = |\mathcal{DES}(X, mv(r_X))| - des(X\overline{Y}, mv(r_{X\overline{Y}}))$. We then have a similar behavior as emphasized in Propositions 1 and 2:
$|\mathcal{DES}(X \rightarrow \vee Y, mv(r_{X \rightarrow \vee Y}))| = \mathcal{F}(X \rightarrow \vee Y, r) - \mathcal{F}(X \rightarrow \vee Y, mv(r))$.

Moreover, an object is deactivated for an association $X \rightarrow \vee Y$ if it is deactivated for X, or if it contains X but every attribute of Y is missing. Denoting $\mathcal{DES}(\wedge Y, mv(r_{X \rightarrow \vee Y})_X)$ for these objects, we have $|\mathcal{DES}(X \rightarrow \vee Y, mv(r_{X \rightarrow \vee Y})) = |\mathcal{DES}(X, mv(r_{X \rightarrow \vee Y}))| + |\mathcal{DES}(\wedge Y, mv(r_{X \rightarrow \vee Y})_X)|$.

3.3 k-freeness definition and correction in incomplete databases

With the help of the deactivation of the incomplete objects, the frequency of $X\overline{Y}$ in r can be bounded by two quantities which are computed in $mv(r)$:

Property 1. $\mathcal{F}(X\overline{Y}, mv(r)) - |\mathcal{DES}(\wedge Y, (mv(r))_X)| \leq \mathcal{F}(X\overline{Y}, r) \leq \mathcal{F}(X\overline{Y}, mv(r)) + |\mathcal{DES}(X, mv(r))|$.

Proof. Proposition 2 says that $\mathcal{F}(X\overline{Y}, r) = \mathcal{F}(X\overline{Y}, mv(r)) + des(X\overline{Y}, mv(r_{X\overline{Y}}))$. The deactivation of an association allows to write $des(X\overline{Y}, mv(r_{X\overline{Y}})) = |\mathcal{DES}(X, mv(r_X))| - |\mathcal{DES}(X \rightarrow \vee Y, mv(r_{X \rightarrow \vee Y}))|$. On one hand, we have the upper bound $des(X\overline{Y}, mv(r_{X\overline{Y}})) \leq |\mathcal{DES}(X, mv(r_X))|$, and when avoiding the restriction on the deactivation database, $des(X\overline{Y}, mv(r_{X\overline{Y}})) \leq |\mathcal{DES}(X, mv(r))|$. On the other hand, we break up $des(X\overline{Y}, mv(r_{X\overline{Y}})) = |\mathcal{DES}(X, mv(r_X))| - (|\mathcal{DES}(X, mv(r_{X \rightarrow \vee Y}))| + |\mathcal{DES}(\wedge Y, mv(r_{X \rightarrow \vee Y})_X)|) = (|\mathcal{DES}(X, mv(r_X))| - |\mathcal{DES}(X, mv(r_{X \rightarrow \vee Y}))|) - |\mathcal{DES}(\wedge Y, mv(r_{X \rightarrow \vee Y})_X)|$. The difference $|\mathcal{DES}(X, mv(r_X))| - |\mathcal{DES}(X, mv(r_{X \rightarrow \vee Y}))|$ is positive so we have the lower bound $des(X\overline{Y}, mv(r_{X\overline{Y}})) \geq |\mathcal{DES}(\wedge Y, mv(r_{X \rightarrow \vee Y})_X)|$. Without the restriction on the deactivation database, $des(X\overline{Y}, mv(r_{X\overline{Y}})) \geq |\mathcal{DES}(\wedge Y, mv(r)_X)|$.

The k-freeness property can be defined in incomplete databases with the bounds for the frequency of $X\overline{Y}$.

Definition 8 **(k-freeness in incomplete databases).**

- *A pattern Z is k-free in $mv(r)$ and we denote $kFree(Z, mv(r))$ if and only if* $\forall XY = Z,\ |Y| \leq k,\ \mathcal{F}(X\overline{Y}, mv(r)) - |\mathcal{DES}(\wedge Y, (mv(r))_X)| > 0$.
- *A pattern Z is k-dependent in $mv(r)$ and we denote $kDepdt(Z, r)$ if and only if* $\exists XY = Z,\ |Y| \leq k,\ \mathcal{F}(X\overline{Y}, mv(r)) + |\mathcal{DES}(X, mv(r))| = 0$.

k-freeness and k-dependence are independently introduced. Section 3.4 will justify this distinction because these definitions are not reverse, due to the missing values.

Let us first note that, in a complete database, our definition of the k-freeness is *compatible* with the classical Definition 3. In this case, the set of deactivated objects is empty when there is no missing values. It is an important point in order to design algorithms which work indifferently on complete or incomplete contexts.

The k-freeness in an incomplete database is linked to this in a complete database with the important following theorem:

Theorem 1 (k-freeness correction). *Let r' be an incomplete database and $mv()$ a missing values modeling operator. For every complete database r such that $mv(r) = r'$ and every pattern Z,*
- $kFree(Z, r') \Longrightarrow kFree(Z, r)$;
- $kDepdt(Z, r') \Longrightarrow \neg kFree(Z, r)$.
The k-free patterns of r' are k-correct.

Proof. Property 1 shows that $\mathcal{F}(X\overline{Y}, r)$ is bounded by $\mathcal{F}(X\overline{Y}, r') - |\mathcal{DES}(\wedge Y, r'_X)|$ and $\mathcal{F}(X\overline{Y}, r') + |\mathcal{DES}(X, r')|$. If the lower bound is strictly positive, $\mathcal{F}(X\overline{Y}, r)$ is also strictly positive then non null and the pattern is k-free in r. If the upper bound is null, $\mathcal{F}(X\overline{Y}, r)$ is null and the pattern is not k-free in r.

Computed with Definition 8, the k-free patterns are then k-correct, i.e. they are k-free in every database completion. In [21,22], this correction is shown for the particular case when $k = 1$. These definitions of the k-freeness and the k-dependence allow to compute properties which are true in every completion: our definitions are **correct**. They are also *complete* because they characterize all k-free patterns in every completion:

Theorem 2 (k-freeness completeness). *Let r' be an incomplete database. If Z is k-free in every complete database r such that there exists a modeling operator $mv()$ with $mv(r) = r'$, then Z is k-free in r'.*

Proof. Suppose the converse, i.e. let Z be k-free in every database r such that $mv(r) = r'$ but non k-free in r'. $\exists XY = Z \mid \mathcal{F}(X\overline{Y}, r') - \mathcal{DES}(\wedge Y, r'_X) \leq 0$. Let r_0 be the database stemming from r' with replacing each missing value by an absent value, then $mv(r_0) = r'$. In r_0, the deactivation is null because r_0 is complete, and the computation of $\mathcal{F}(X\overline{Y}, r_0)$ gives the same result as in r' where it is done with the frequencies of the present attributes. $\mathcal{F}(X\overline{Y}, r_0)$ is then null and Z is not k-free is r_0 : contradiction.

In an incomplete database, every computed k-free pattern is k-correct and every pattern which is k-free in every completion of the database is covered by this definition.

3.4 Properties of the k-freeness in incomplete databases

The k-freeness and the k-dependency are not complementary: some patterns will be neither k-free nor k-dependent because it is sometimes impossible to decide if they are present or not in an object. The table below details the computation of 1-freeness for the pattern a_4a_7:

X	Y	$\mathcal{F}(X\overline{Y}, mv(r))$	$\lvert\mathcal{DES}(\wedge Y, mv(r)_X)\rvert$	$\lvert\mathcal{DES}(X, mv(r))\rvert$	1-free?	1-dependent?
a_4	a_7	1	1	1	$1-1 \not> 0$: no	$1+1 \neq 0$: no
a_7	a_4	1	1	1	$1-1 \not> 0$: no	$1+1 \neq 0$: no

We now give a vital property for designing k-free patterns mining algorithms. It refers to the (anti)-monotonicity of the k-freeness or dependency. The k-freeness does not satisfy a property of (anti-)monotonicity, but Theorem 3 indicates that the k-dependency is monotone.

Theorem 3 (Monotonicity of the k-dependency property). *The k-dependency property is monotone, i.e. for all patterns Z and every database r', $Z \subseteq Z' \Rightarrow (kDepdt(Z, r') \Rightarrow kDepdt(Z', r'))$.*

Proof. Let Z be a k-dependent pattern. $\exists XY = Z$, $\mathcal{F}(X\overline{Y}, mv(r)) + |\mathcal{DES}(X, mv(r))| = 0$ or $\mathcal{F}(X\overline{Y}, mv(r)) = 0$ and $|\mathcal{DES}(X, mv(r))| = 0$. $\mathcal{F}(X\overline{Y}, mv(r)) = 0$ means that for all object $o \in \mathcal{O}$, $X \subseteq o \Rightarrow Y \cap o \neq \emptyset$. *A fortiori*, $X \subseteq o \Rightarrow aY \cap o \neq \emptyset$ for all $a \in \mathcal{A}$, then $\mathcal{F}(X\overline{aY}, mv(r)) = 0$. By induction on all attributes of $Z'\backslash Z$, one deduces that Z' is also k-dependent.

With this result, the framework of the level-wise algorithms can be used with the negation of the k-dependency constraint, and we have written the `MV-k-miner` prototype. The full algorithm is provided in [12].

4 Experiments on UCI benchmarks

We now reproduce the experiments described in Section 2.2 and measure the number of 3-free patterns computed with `MVminer` in $mv(r)$, compared to r. Results are reported in Figure 3. Due to the lack of space, only `solar-flare` and `zoo` are detailed. In the other datasets, the same trends appear.

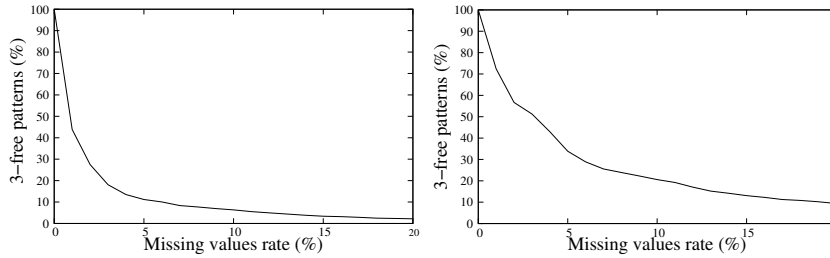


Fig. 3. Proportion of 3-free patterns in $mv(r)$ (r gives 100 %).

As expected, the number of patterns recovered by our method decreases according to the number of missing values. Indeed, each pattern is k-correct or k-free in every complete dataset, whose number is exponential in the number of missing values. But `MV-k-miner` computes only k-correct patterns. While data mining is known to produce a huge number of patterns, their correctness is essential. Missing values damages are then avoided and this result opens the way for the uses of k-free patterns mentioned in Section 2.1.

5 Conclusion

We proposed a definition for the k-free property in an incomplete database. Thanks to this new definition, the mined patterns are k-correct, or k-free in every completion of the database: this avoids damages due to missing values. Our perspectives address the classification with generalized associations and its application to missing values imputation.

References

1. Grzymala-Busse, J., Hu, M.: A comparison of several approaches to missing attribute values in data mining. In: RSCTC '00: Revised Papers from the Second International Conference on Rough Sets and Current Trends in Computing, London, UK. (2001) 378–385
2. Calders, T., Goethals, B.: Minimal k-free representations of frequent sets. In: Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD'03). (2003) 71–82
3. Calders, T., Goethals, B.: Mining all non-derivable frequent itemsets. In: Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD'02). (2002)
4. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules. In: Intl. Conference on Very Large Data Bases (VLDB'94), Santiago de Chile, Chile. (1994) 487–499
5. Bykowski, A., Rigotti, C.: A condensed representation to find frequent patterns. In: ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems. (2001) 267–273
6. Blake, C., Merz, C.: UCI repository of machine learning databases (1998)
7. Bastide, Y., Taouil, R., Pasquier, N., Stumme, G., Lakhal, L.: Mining minimal non-redundant association rules using frequent closed itemsets. In: International Conference on Deductive and Object Databases (DOOD'00). (2000) 972–986
8. Zaki, M.: Generating non-redundant association rules. In: ACM SIGKDD international conference on Knowledge discovery and data mining, Boston, USA. (2000) 34–43
9. Rioult, F.: Extraction de connaissances dans les bases de données comportant des valeurs manquantes ou un grand nombre d'attributs. PhD thesis, Université de Caen Basse-Normandie, France (2005)
10. Gunopulos, D., Mannila, H., Khardon, R., Toivonen, H.: Data mining, hypergraph transversals, and machine learning. In: PODS'97. (1997)
11. Mannila, H., Toivonen, H.: Levelwise search and borders of theories in knowledge discovery. Data Mining and Knowledge Discovery **1**(3) (1997) 241–258
12. Rioult, F., Crèmilleux, B.: Extraction de propriétés correctes dans des bases de données incomplétes. In: Actes de la conférence francophone sur l'apprentissage automatique (CAp'06), Trégastel, France, Presses Universitaires de Grenoble (2006) 347–362
13. Antonie, M.L., Zaïane, O.: An associative classifier based on positive and negative rules. In: DMKD'04. (2004)
14. Antonie, M.L., Zaïane, O.: Mining positive and negative association rules: An approach for confined rules. In: PKDD'04. (2004) 27–38
15. Jaroszewicz, S., Simovici, D.: Support approximations using bonferroni-type inequalities. In: Principles of Data Mining and Knowledge Discovery (PKDD'02). (2002) 212–224
16. Dyreson, C.E.: A Bibliography on Uncertainty Management in Information Systems. In: Uncertainty Management in Information Systems. Kluwer Academic Publishers (1997)
17. Levene, M., Loizou, G.: Database design for incomplete relations. ACM Transactions on Database Systems **24**(1) (1999) 80–126
18. Ragel, A., Crémilleux, B.: Mvc - a preprocessing method to deal with missing values. Knowledge-Based Systems **12**(5-6) (1999) 285–291
19. Nayak, J., Cook, D.: Approximate association rule mining. In: Florida Artificial Intelligence Research Symposium, Key West, Florida, USA. (2001) 259–263
20. Jami, S., Jen, T., Laurent, D., Loizou, G., Sy, O.: Extraction de règles d'association pour la prédiction de valeurs manquantes. In: Colloque Africain sur la Recherche en Informatique (CARI). (2004)
21. Rioult, F., Crémilleux, B.: Condensed representations in presence of missing values. In: Symposium on Intelligent Data Analysis, Berlin, Germany. (2003) 578–588
22. Rioult, F., Crémilleux, B.: Représentation condensée en présence de valeurs manquantes. In: XXIIè congrès Inforsid, Biarritz, France. (2004) 301–317

Efficient Mining under Flexible Constraints through Several Datasets

Arnaud Soulet, Jiří Kléma, and Bruno Crémilleux

GREYC, CNRS - UMR 6072, Université de Caen
Campus Côte de Nacre, F-14032 Caen Cédex France
{Forename.Surname}@info.unicaen.fr

Abstract. Mining patterns under many kinds of constraints is a key point to successfully get new knowledge. In this paper, we propose an efficient new algorithm MUSIC-DFS which soundly and completely mines patterns with various constraints from large data and takes into account external data represented by several heterogeneous datasets. Constraints are freely built of a large set of primitives and enable to link the information scattered in various knowledge sources. Efficiency is achieved thanks to a new closure operator providing an interval pruning strategy applied during the depth-first search of a pattern space. A genomic case study shows both the effectiveness of our approach and the added-value of background knowledge such as free texts or gene ontologies in discovery of meaningful patterns.

1 Introduction

In current scientific, industrial or business data mining applications, the critical need is not to generate data, but to derive knowledge from huge and heterogeneous datasets produced at high throughput. Putting all this data together has become a pressing need for developing environments and tools able to explore and discover new highly valuable knowledge. This involves different challenges, like designing efficient tools to tackle a large amount of data and the discovery of patterns of a potential user's interest through several datasets. Constraints provide a focus on the most promising knowledge by reducing the number of extracted patterns to those of a potential interest given by the user. Furthermore, when constraints can be pushed deep inside the mining algorithm, performance is improved, making the mining task computationally feasible and resulting in a human-workable output.

This paper addresses the issue of efficient mining under flexible constraints from large binary data combined with several heterogeneous external datasets synthetizing background knowledge (BK). Large datasets are characterized mainly by a large number of columns (i.e., items). This characteristic often encountered in a lot of domains (e.g., bioinformatics, text mining) represents a remarkable challenge. Usual algorithms show difficulties in running on this kind of data due to the exponential search space growth with the number of items. Known level-wise algorithms can fail in mining frequent or constrained patterns in such data [5]. On top of that, the user often would like to integrate BK in the mining process in order to focus on the most plausible patterns consistent with pieces of existing

knowledge. BK is available in relational and literature databases, ontological trees and other sources. Nevertheless, mining in a heterogeneous environment allowing a large set of descriptions at various levels of detail is highly non-trivial. This paper solves the problem by pushing user-defined constraints that may stem both from the mined binary data and the BK summarized in similarity matrices or textual files.

The contribution of this paper is twofold. First we provide a new algorithm Music-dfs which soundly and completely mines constrained patterns from large data while taking into account external data (i.e., several heterogeneous datasets). Except for specific constraints for which tricks like the transposition of data [8, 5] or the use of the extension [4] can be used, levelwise approaches cannot tackle large data due to the huge number of candidates. On the contrary, Music-dfs is based on a depth first search strategy. The key idea is to use a new closure operator enabling an efficient interval pruning for varied constraints (see Section 3). In [3], the authors also benefit from intervals to prune the search space, but their approach is restricted to the conjunction of one monotone constraint and one anti-monotone constraint. The output of Music-dfs is an interval condensed representation: each pattern satisfying the given constraint appears once in the collection of intervals only. Second, we provide a generic framework to mine patterns with a large set of constraints based on several heterogeneous datasets like texts or similarity matrices. It is a way to take into account the BK. Section 4 depicts a genomic case study. The biological demands require to mine the expression data with constraints concerning complex relations represented by free texts and gene ontologies. The discovered patterns are likely to encompass interesting and interpretable knowledge.

This paper differs for a double reason from our work in [11]. First, the framework is extended to external data. Second, Music-dfs is deeply different from the prototype used in [11]: Music-dfs integrates primitives to tackle external data and thanks to its strategy to prune the search space (new interval pruning based on prefix-free patterns, see Section 3), it is able to mine large data. Section 4 demonstrates the practical effectiveness of Music-dfs in a genomic case study and shows that other prototypes (including the prototype presented in [11]) fail. To the best of our knowledge, there is no other constraint-based tool to efficiently discover patterns from large data under a broad set of constraints linking the information distributed in various knowledge sources.

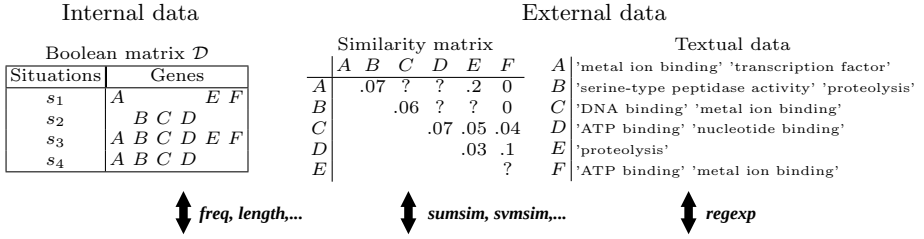
This paper is organized as follows. Section 2 defines our framework to mine patterns satisfying constraints defined over several kinds of datasets. In Section 3, we present the theoretical essentials that underlie the efficiency of Music-dfs and we provide its main features. Experiments showing the efficiency of Music-dfs and the cross-fertilization between several sources of information related to the genomic area are given in Section 4.

2 Defining Constraints Through Several Datasets

Usual data-mining tasks can rarely be represented by a single binary dataset. Often it is necessary to connect knowledge scattered in several heterogeneous sources. In

constraint-based mining, the constraints should effectively link different datasets and knowledge types. For instance, in the domain of genomics, biologists are interested in constraints both on synexpression groups and common characteristics of the genes and/or biological situations being concerned. Such constraints require to tackle both transcriptome data (often provided in a transactional format) and literature databases. This section presents our framework (and the declarative language) enabling the user to set varied and meaningful constraints.

Let us start with a toy genomic example given in Figure 1. Firstly, the mining context is made up of a boolean dataset also called internal data (or transcriptome data) where the items correspond to genes, the transactions represent biological situations. Secondly, external data (a similarity matrix and a textual dataset) are considered. They summarize the BK that contains various information on items (i.e., genes). This knowledge is transformed into a similarity matrix and a set of texts. Each field of the triangular matrix $s_{ij} \in [0, 1]$ gives a similarity measure between the items i and j. The textual dataset provides a description of genes. Each row of this dataset contains a list of phrases characterizing the given gene. The mined patterns are composed of items of the internal data, the external data are used to further specify constraints in order to focus on meaningful patterns. In other words, the constraints may stem from all the datasets (see the example of q in Figure 1, Section 4 provides another q').



$$
\begin{aligned}
q(X) \equiv\ & freq(X) \times length(X) \geq 24 && \text{(a)}\\
& \wedge length(regexp(X, '*ribosom*', \texttt{TEXT\_terms})) \leq 1 && \text{(b)}\\
& \wedge svsim(X, \texttt{TEXT})/(svsim(X, \texttt{TEXT}) + mvsim(X, \texttt{TEXT})) \geq 0.7 && \text{(c)}\\
& \wedge sumsim(X, \texttt{TEXT})/svsim(X, \texttt{TEXT}) \geq 0.025 && \text{(d)}
\end{aligned}
$$

Fig. 1. Example of a toy (genomic) mining context and a constraint.

Let $\mathcal{I}$ be a set of items, a pattern is a non-empty subset of $\mathcal{I}$. $\mathcal{D}$ is a boolean matrix composed of patterns usually called transactions. The constraint-based mining task aims to discover all the patterns present in $\mathcal{D}$ and satisfying a constraint q. A pattern X is present in $\mathcal{D}$ whenever it is included in one transaction of $\mathcal{D}$ at least.

The strength and originality of our framework lies also in its flexibility. Constraints are freely built of a large set of primitives representing an integrative, iterative and rich query language. Table 1 provides the meaning of the primitives involved in q and also the constraints used in Section 4. As primitives on external data are derived from different datasets, the dataset makes another parameter of the primitive (it is not present in Table 1 to alleviate the writing).

The first part (a) of q addresses the internal data and means that the biologist is interested in patterns having a satisfactory size – a *minimal area*. Indeed, $area(X) = freq(X) \times length(X)$ is the product of the frequency of X and its length and means that the pattern must cover a minimum number of situations and contain a minimum number of genes. The other parts deal with the external data: (b) is used to discard ribosomal patterns (one gene exception per pattern is allowed), (c) avoids patterns with prevailing items of an unknown function and (d) is to ensure a minimal average gene similarity. Table 1 also indicates the values of these primitives in the context of Figure 1. Our framework supports a large set of primitives, other examples of primitives are $\{\wedge, \vee, \neg, <, \leq, \subset, \subseteq, +, -, \times, /, sum, max, min, \cup, \cap, \setminus\}$. The only property which is required on the primitives to belong to our framework is a property of monotonicity according to each variable of a primitive [11]. The constraints of this framework are called *primitive-based constraints*. Let us recall that the primitives and the constraints defined in [11] only address one boolean data set.

primitives		values
Boolean matrix		
$freq(X)$	frequency of X	$freq(ABC) = 2$
$length(X)$	length of X	$length(ABC) = 3$
Textual data		
$regexp(X, RE)$	items of X whose associated phrases match the regular expression RE	$regexp(ABC, '*ion*') = AC$
Similarity matrix		
$sumsim(X)$	similarity sum over the set of item pairs of X	$sumsim(ABC) = 0.13$
$svsim(X)$	number of stated item pairs belonging to X	$svsim(ABC) = 2$
$mvsim(X)$	number of missing item pairs belonging to X	$mvsim(ABC) = 1$
$insim(X, min, max)$	number of item pairs whose similarity lies between min and max	$insim(ABC, 0.07, 1) = 1$

Table 1. Examples of primitives and their values in the data mining context of Figure 1.

3 MUSIC-DFS Tool

This section presents the MUSIC-DFS tool (Mining with a User-SpecifIed Constraint, Depth-First Search approach) which benefits from the primitive-based constraints presented in the previous section. Efficiency is achieved thanks to the exploitation of the primitive and constraint properties. We start by giving the key idea of the safe pruning process based on intervals.

3.1 Main features of the interval pruning

The pruning process performed by MUSIC-DFS is based on the key idea to exploit properties of the monotonicity of the primitives (see Section 2) on the bounds of intervals to prune them. This new kind of pruning is called *interval pruning*. Given two patterns $X \subseteq Y$, the interval $[X, Y]$, also called sub-algebra or sub-lattice, corresponds to the set $\{Z \subseteq \mathcal{I} \mid X \subseteq Z \subseteq Y\}$. Figure 2 depicts an example with the interval $[AB, ABCD]$ and the values of the primitives *sumsim* and *svsim*.

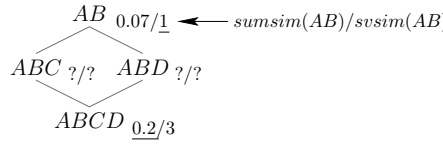


Fig. 2. Illustration of the interval pruning.

Assume the constraint $sumsim(X)/svsim(X) \geq 0.25$. As the values associated to the similarities are positive, $sumsim(X)$ is an increasing function according X. Thus $sumsim(ABCD)$ is the highest *sumsim* value for the patterns in $[AB, ABCD]$. Similarly, all the patterns of this interval have a higher $svsim(X)$ value than $svsim(AB)$. Thereby, each pattern in $[AB, ABCD]$ has its average similarity lower or equal than $sumsim(ABCD)/svsim(AB) = 0.2/1$. As this fraction does not exceed 0.25, no pattern of $[AB, ABCD]$ can satisfy the constraint and this interval can be pruned. We say that this pruning is *negative* because no pattern satisfies the constraint. In the same way, if the values of proper combinations of the primitives on the bounds of an interval $[X, Y]$ show that all the patterns in $[X, Y]$ satisfy the constraint, then $[X, Y]$ is also pruned and this pruning is named *positive*. For instance, assuming that $sumsim(AB)/svsim(ABCD) \geq 0.02$, then all the patterns in $[AB, ABCD]$ satisfy the constraint.

In a more formal way, this approach is performed by two interval pruning operators $\lfloor . \rfloor$ and $\lceil . \rceil$ introduced in [11] (but only for primitives handling boolean data). The main idea of these operators is to recursively decompose the constraint to benefit from the monotone properties of the primitives and then to safely negatively or positively prune intervals as depicted above. This process works straightforwardly with all the primitives no matter what kind of dataset they regard. This highlights the generic properties of our framework, as well the feature of pushing all the parts of the constraint q into the mining step. The next section indicates how the intervals are built.

3.2 Interval condensed representation

As indicated in Section 1, levelwise algorithms are not suitable to mine datasets with a large number of items due to the huge number of candidates growing exponentially according to the number of items. We adopt a depth-first search strategy instead of enumerating the candidate patterns and avoiding subsequent memory failures. We introduce a new and specific closure operator based on a prefix ordering relation $\preceq$. We show that this closure operator is on the core of the interval condensed representation (Theorem 1) leading to an efficient pruning strategy covering in depth the search space.

The prefix ordering relation $\preceq$ starts from an arbitrary order over items $A < B < C < \ldots$ as done in [10]. We say that an ordered pattern $X = x_1 x_2 \ldots x_n$ (i.e., $\forall i < j$, we have $x_i < x_j$) is a prefix of an ordered pattern $Y = y_1 y_2 \ldots y_m$

and note $X \preceq Y$ iff we have $n \leq m$ and $\forall i \in \{1, \ldots, n\},\ x_i = y_i$. For instance, $AD \not\preceq ADC$ because $ADC = ACD$ and AD is not a prefix of ACD.

Definition 1 (Prefix-closure). *The prefix-closure of a pattern X, denoted $\mathbf{cl}_{\preceq}(X)$, is the pattern $\{a \in \mathcal{I} | \exists Y \subseteq X$ such that $Y \preceq Y \cup \{a\}$ and $freq(Ya) = freq(Y)\}$.*

The pattern $\mathbf{cl}_{\preceq}(X)$ gathers together the items occurring in all the transactions containing $Y \subseteq X$ such that Y is a prefix of $Y \cup \{a\}$. The fixed points of operator $\mathbf{cl}_{\preceq}$ are named the *prefix-closed patterns*. Let us illustrate this definition on our running example (cf. Figure 1). The pattern ABC is not a prefix-closed pattern because ABC is a prefix of $ABCD$ and $freq(ABCD) = freq(ABC)$. On the contrary, $ABCD$ is prefix-closed. We straightforwardly deduce that any pattern and its prefix-closure have the same frequency. For instance, as $\mathbf{cl}_{\preceq}(ABC) = ABCD$, $freq(ABC) = freq(ABCD) = 2$. Now we show the property of closure of $\mathbf{cl}_{\preceq}$:

Property 1 (Closure operator) *The prefix-closure operator $\mathbf{cl}_{\preceq}$ is a closure operator.*

Proof. *Extensivity:* Let X be a pattern and $a \in X$. We have $\{a\} \subseteq X$ and obviously, $a \preceq a$ and $freq(a) = freq(a)$. Then, we obtain that $a \in \mathbf{cl}_{\preceq}(X)$ and $\mathbf{cl}_{\preceq}$ is extensive. *Isotony:* Let $X \subseteq Y$ and $a \in \mathbf{cl}_{\preceq}(X)$. There exists $Z \subseteq X$ such that $Z \preceq Za$ and $freq(Za) = freq(Z)$. As we also have $Z \subseteq Y$ (and $freq(Za) = freq(Z)$), we obtain that $a \in \mathbf{cl}_{\preceq}(Y)$ and conclude that $\mathbf{cl}_{\preceq}(X) \subseteq \mathbf{cl}_{\preceq}(Y)$. *Idempotency:* Let X be a pattern. Let $a \in \mathbf{cl}_{\preceq}(\mathbf{cl}_{\preceq}(X))$. There exists $Z \subseteq \mathbf{cl}_{\preceq}(X)$ such that $freq(Za) = freq(Z)$ with $Z \preceq Za$. As $Z \subseteq \mathbf{cl}_{\preceq}(X)$, for all $a_i \in Z$, there is $Z_i \subseteq X$ such that $freq(Z_i a_i) = freq(Z_i)$ with $Z_i \preceq Z_i a_i$. We have $\bigcup_i Z_i \preceq \bigcup_i Z_i a$ and $freq(\bigcup_i Z_i) = freq(\bigcup_i Z_i a)$ (because $freq(\bigcup_i Z_i) = freq(Z)$). As the pattern $\bigcup_i Z_i \subseteq X$, a belongs to $\mathbf{cl}_{\preceq}(X)$ and then, $\mathbf{cl}_{\preceq}$ is idempotent. □

Property 1 is important because it enables to infer results requiring the properties of a closure operator. First, this new prefix-closure operator designs *equivalence classes* through the lattice of patterns. More precisely, two patterns X and Y are equivalent iff they have the same prefix-closure (i.e., $\mathbf{cl}_{\preceq}(X) = \mathbf{cl}_{\preceq}(Y)$). Of course, as $\mathbf{cl}_{\preceq}$ is idempotent, the maximal pattern (w.r.t. $\subseteq$) of a given equivalence class of X corresponds to the prefix-closed pattern $\mathbf{cl}_{\preceq}(X)$. Conversely, we call *prefix-free patterns* the minimal patterns (w.r.t. $\subseteq$) of equivalence classes. Second, closure properties enable to prove that the prefix-freeness is an anti-monotone constraint (see Property 2 in the next section).

Contrary to the equivalence classes defined by the Galois closure [2, 9], equivalence classes provided by $\mathbf{cl}_{\preceq}$ have a unique prefix-free pattern. This allows to prove that a pattern belongs to one interval only and provides the important result on the interval condensed representation (cf. Theorem 1). This result cannot be achieved without the new closure operator. Lemma 1 indicates that any equivalence class has a unique prefix-free pattern:

Lemma 1 (Prefix-freeness operator). *Let X be a pattern, there exists an unique minimal pattern (w.r.t. $\subseteq$), denoted $\mathbf{fr}_{\preceq}(X)$, in its equivalence class.*

Proof. Supposing that X and Y are two minimal patterns of the same equivalence class: we have $\mathbf{cl}_{\preceq}(X) = \mathbf{cl}_{\preceq}(Y)$. As X and Y are different, there exists $a \in X$ such that $a \notin Y$ and $a \leq min_{\preceq}\{b \in Y \backslash X\}$ (or we invert X and Y). As X is minimal, no pattern $Z \subseteq X \cap Y$ satisfies that $Z \preceq Za$ and $freq(Za) = freq(Z)$. Besides, for all Z such that $Y \cap X \subset Z \subset Y$, we have $Z \not\preceq Za$ because a is smaller than any item of $Y \backslash X$. So, a does not belong to $\mathbf{cl}_{\preceq}(Y)$ and then, we obtain that $\mathbf{cl}_{\preceq}(X) \neq \mathbf{cl}_{\preceq}(Y)$. Thus, we conclude that any equivalence class exactly contains one prefix-free pattern. □

Lemma 1 means that the operator $\mathbf{fr}_{\preceq}$ links a pattern X to the minimal pattern of its equivalence class, i.e. $\mathbf{fr}_{\preceq}(X)$. X is prefix-free iff $\mathbf{fr}_{\preceq}(X) = X$. Any equivalence class corresponds to an interval delimited by one prefix-free pattern and its prefix-closed pattern (i.e., $[\mathbf{fr}_{\preceq}(X), \mathbf{cl}_{\preceq}(X)]$). For example, AB (resp. $ABCD$) is the prefix-free (resp. prefix-closed) pattern of the equivalence class $[AB, ABCD]$.

Now let us show that the whole collection of the intervals formed by all the prefix-free patterns and their prefix-closed patterns provides an *interval condensed representation* where each pattern X is present only once in the set of intervals.

Theorem 1 (Interval condensed representation). *Each pattern X present in the dataset is included in the interval $[\mathbf{fr}_{\preceq}(X), \mathbf{cl}_{\preceq}(X)]$. Besides, the number of these intervals is less than or equal to the number of patterns.*

Proof. Let X be a pattern and $R = \{[\mathbf{fr}_{\preceq}(X), \mathbf{cl}_{\preceq}(X)] | freq(X) \geq 1\}$. Lemma 1 proves that X is exactly contained in $[\mathbf{fr}_{\preceq}(X), \mathbf{cl}_{\preceq}(X)]$. The latter is unique. As X belongs to R by definition, we conclude that R is a representation of any pattern. Now, the extensivity and the idempotency of prefix-closure operator $\mathbf{cl}_{\preceq}$ ensure that $|R| \leq |\{X \subseteq \mathcal{I}$ such that $freq(X) \geq 1\}|$. Thus, Theorem 1 is right. □

In the worst case the size of the condensed representation is the number of patterns (each pattern is its own prefix-free and its own prefix-closed pattern). But, in practice, the number of intervals is low compared to the number of patterns (in our running example, only 23 intervals sum up the 63 present patterns).

The condensed representation highlighted by Theorem 1 differs from the condensed representations of frequent patterns based on the Galois closure [2, 9]: in this last case, intervals are described by a free (or key) pattern and its Galois closure and a frequent pattern may appear in several intervals. We claim that the presence of a pattern in a single interval brings meaningful advantages: the mining is more efficient because each pattern is tested at most once. This property improves the synthesis of the output of the mining process and facilitates its analysis by the end-user. The next section shows that by combining this condensed representation and the interval pruning operators, we get an interval condensed representation of primitive-based constrained patterns.

3.3 Mining primitive-based constraints in large datasets

When running, MUSIC-DFS enumerates all the intervals sorted in a lexicographic order and checks whether they can be pruned as proposed in Section 3.1. The enumeration benefits from the anti-monotonicity property of the prefix-freeness

(cf. Property 2). The memory requirements grow only linearly with the number of items and the number of transactions.

Property 2 *The prefix-freeness is an anti-monotone constraint (w.r.t. $\subseteq$).*

The proof of Property 2 is very similar to those of the usual freeness [2, 9]. In other words, the anti-monotonicity ensures us that once we know that a pattern is not prefix-free, any superset of this pattern is not prefix-free anymore [1, 7]. Algorithms 1 and 2 give the sketch of MUSIC-DFS.

Algorithm 1 GLOBALSCAN

Input: A prefix-pattern X, a primitive based constraint q and a dataset $\mathcal{D}$
Output: Interval condensed representation of constrained patterns having X as prefix
1: **if** $\neg PrefixFree(X)$ **then return** $\emptyset$ // *anti-monotone pruning*
2: **return** LOCALSCAN$([X, \mathbf{cl}_{\preceq}(X)], q, \mathcal{D})$ // *local mining*
$\cup \bigcup\{$GLOBALSCAN$(Xa, q, \mathcal{D}) | a \in \mathcal{I} \wedge a \geq \max_{\leq} X\}$ // *recursive enumeration*

Algorithm 2 LOCALSCAN

Input: An interval $[X, Y]$, a primitive based constraint q and a dataset $\mathcal{D}$
Output: Interval condensed representation of constrained patterns of $[X, Y]$
1: **if** $\lfloor q \rfloor \langle X, Y \rangle$ **then return** $\{[X, Y]\}$ // *positive interval pruning*
2: **if** $\neg \lceil q \rceil \langle X, Y \rangle$ **then return** $\emptyset$ // *negative interval pruning*
3: **if** $q(X)$ **then return** $[X, X] \cup \bigcup\{$LOCALSCAN$([Xa, \mathbf{cl}_{\preceq}(Xa)], q, \mathcal{D}) | a \in Y \backslash X\}$
4: **return** $\bigcup\{$LOCALSCAN$([Xa, \mathbf{cl}_{\preceq}(Xa)], q, \mathcal{D}) | a \in Y \backslash X\}$ // *recursive division*

MUSIC-DFS scans the whole search space by running GLOBALSCAN on each item of $\mathcal{I}$. GLOBALSCAN recursively performs a depth-first search and stops whenever a pattern is not prefix-free (Line 1, GLOBALSCAN). For each prefix-free pattern X, it computes its prefix-closed pattern and builds $[X, \mathbf{cl}_{\preceq}(X)]$ (Line 2, GLOBALSCAN). Then, LOCALSCAN tests this interval by using the operators $\lfloor . \rfloor$ and $\lceil . \rceil$ informally presented in Section 3.1. If the interval pruning can be performed, the interval is selected (positive pruning, Line 1 from LOCALSCAN) or rejected (negative pruning, Line 2 from LOCALSCAN). Otherwise, the interval is explored by recursively dividing it (Line 3 or 4 from LOCALSCAN). The decomposition of the intervals is done so that each pattern is considered only once. The next theorem provides the correctness of MUSIC-DFS:

Theorem 2 (Correctness). MUSIC-DFS *mines soundly and completely all the patterns satisfying q by means of intervals.*

Proof. Property 2 ensures us that MUSIC-DFS enumerates all the interval condensed representation. Thereby, any pattern is considered (Theorem 1) individually or globally with the safe pruning stemmed from to the interval pruning (see Section 3.1). □

4 Mining Constrained Patterns from Genomic Data

This section depicts the effectiveness of our approach on a genomic case study. We experimentally show two results. First, the usefulness of the interval pruning strategy of MUSIC-DFS (the other prototypes fail for such large data, cf. Section 4.2). Second, BK enables to automatically focus on the most plausible candidate patterns (cf. Section 4.3). This underlines the need to mine constrained patterns by taking into account external data.

4.1 Gene expression data and background knowledge

In this experiment we deal with the SAGE (Serial Analysis of Gene Expression) [12] human expression data downloaded from the NCBI website (`www.ncbi.nlm.nih.gov`). The final binary dataset contains 11082 genes tested in 207 biological situations, each gene can be either over-expressed in the given situation or not. The biological details regarding gene selection, mapping and binarization can be seen in [6].

BK available in literature databases, biological ontologies and other sources is used to help to focus automatically on the most plausible candidate patterns. We have experimented with the gene ontology (GO) and free-text data. First, the available gene databases were automatically searched and the records for each gene were built (around two thirds of genes have non-empty records, there is no information available for the rest of them). Then, various similarity metrics among the gene records were proposed and calculated. The gene records were also simplified to get a condensed textual description. The details on text mining, gene ontologies and similarities are in [6].

4.2 Efficiency of MUSIC-DFS

Let us show the necessity of the depth-first search and usefulness of the interval pruning strategy of MUSIC-DFS. All the experiments were conducted on a 2.2 GHz Xeon processor with Linux operating system and 3GB of RAM memory.

The first experiment highlights the importance of the depth-first search. We consider the constraint addressing patterns having an $area \geq 70$ (the minimal area constraint has been introduced in Section 2) and appearing at least 4 times in the dataset. MUSIC-DFS only spends 7sec to extract 212 constrained patterns. In comparison, for the same binary dataset, the levelwise approach[1] presented in [11] fails after 963sec whenever it contains more than 3500 genes. Indeed, the candidate patterns necessary to build the output do not fit in memory.

Comparison with prototypes coming from the FIMI repository (`fimi.cs.helsinki.fi`) shows that efficient implementations like KDCI, LCM (ver. 2), COFI or Borgelt's APRIORI fail with this binary dataset to mine frequent patterns occuring at least 4 times. Borgelt's ECLAT and AFOPT which are depth-first approaches, are able to mine with this frequency constraint. But they require a

[1] We do not use external data because this version does not deal with external data.

post-processing step for other constraints than the frequency (e.g., area, similarity-based constraints).

The next experiment shows the great role of the interval pruning strategy. For this purpose, we compare MUSIC-DFS with its modification that does not prune. The modification, denoted MUSIC-DFS-FILTER, mines all the patterns that satisfy the frequency threshold first, the other primitives are applied in the post-processing step. We use two typical constraints needed in the genomic domain and requiring the external data. These constraints and the time comparison between MUSIC-DFS and MUSIC-DFS-FILTER are given in Figure 3. The results show that post-processing is feasible until the frequency threshold generates reasonable pattern sets. For lower frequency thresholds, the number of patterns explodes and large intervals to be pruned appear. The interval pruning strategy decreases runtime and scales up much better than the comparative version without interval pruning and MUSIC-DFS becomes in the order of magnitude faster.

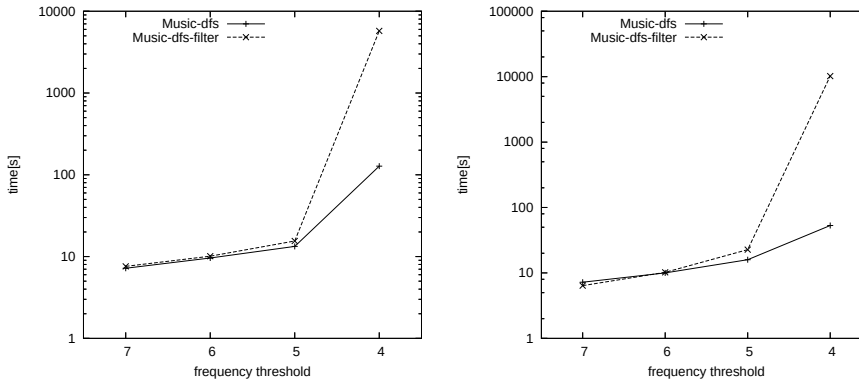


Fig. 3. Efficiency of interval pruning with decreasing frequency threshold. The left image deals with the constraint $freq(X) \geq thres \wedge lenght(X) \geq 4 \wedge sumsim(X)/svsim(X) \geq 0.9 \wedge svsim(X)/(svsim(X)+mvsim(X)) \geq 0.9$. The right image deals with the constraint $freq(X) \geq thres \wedge length(regexp(X,'*ribosom*', \texttt{GO\_terms})) = 0$.

4.3 Use of background knowledge to mine plausible patterns

This genomic case study demonstrates that constraints coming from the BK can reduce the number of patterns, they can express various kinds of interest and the patterns that tend to reappear are likely to be recognized as interesting by an expert. Such an approach requires a tool dealing with external data.

Let us consider all the patterns having a satisfactory size which is translated by the constraint $area \geq 20^2$. We get nearly half a million different patterns that are joined into 37852 intervals. Although the intervals prove to provide a good

[2] This threshold has been settled by statistical analysis of random datasets having the same properties as the original SAGE data. First spurious patterns start to appear for this threshold area.

condensation, the manual search through this set is obviously infeasible as the interpretation of patterns is not trivial and asks for frequent consultations with medical databases. The biologists prefer sets with tens of patterns/intervals only.

Increasing the threshold of the area constraint to get a reasonable number of patterns is rather counter-productive. The constraint $area \geq 75$ led to a small but uniform set of 56 patterns that was flooded by the ribosomal proteins which generally represent the most frequent genes in the dataset. Biologists rated these patterns as valid but uninteresting.

The most valuable patterns expected by biologists – denoted as meaningful or plausible patterns – have non-trivial size containing genes and situations whose characteristics can be generalized, connected, interpreted and thus transformed into knowledge. To get such patterns, constraints based on the external data have to be added to the minimal area constraint just like in the constraint q given in Section 2. It joins the minimal area constraint with background constraints coming from the NCBI textual resources (gene summaries and adjoined PubMed abstracts). There are 46671 patterns satisfying the minimal area constraint (the part (a) of the constraint q), but only 9 satisfy q. This shows the efficiency of reduction of patterns brought by the BK.

A cross-fertilization with other external data is obviously favourable. So, we use the constraint q' which is similar to q, except that the functional Gene Ontology is used instead of NCBI textual resources and a similarity constraint is added (part (e) of q').

$$\begin{aligned} q'(X) \equiv\ & area(X) \geq 24 && \text{(a)}\\ & \wedge length(regexp(X, '*ribosom*', \texttt{GO\_terms})) \leq 1 && \text{(b)}\\ & \wedge svsim(X, \texttt{GO})/(svsim(X, \texttt{GO}) + mvsim(X, \texttt{GO})) \geq 0.7 && \text{(c)}\\ & \wedge sumsim(X, \texttt{GO})/svsim(X, \texttt{GO}) \geq 0.025 && \text{(d)}\\ & \wedge insim(X, 0.5, 1, \texttt{GO})/svsim(X, \texttt{GO}) \geq 0.6 && \text{(e)} \end{aligned}$$

Only 2 patterns satisfy q'. A very interesting observation is that the pattern[3] that was identified by the expert as one of the "nuggets" provided by q is also selected by q'. This pattern can be verbally characterized as follows: it consists of 4 genes that are over-expressed in 6 biological situations, it contains at most one ribosomal gene, the genes share a lot of common terms in their descriptions as well as they functionally overlap, at least 3 of the genes are known (have a non-empty record) and all of the biological situations are medulloblastomas which are very aggressive brain tumors in children. The constraints q and q' demonstrate two different ways to reach a compact and meaningful output that can be easily human surveyed.

5 Conclusion

Knowledge discovery from a large binary dataset supported by heterogeneous BK is an important task. We have proposed a generic framework to mine patterns with

[3] The pattern consists of 4 genes KHDRBS1 NONO TOP2B FMR1 over-expressed in 6 biological situations BM_P019 BM_P494 BM_P608 BM_P301 BM_H275 BM_H876. BM stands for brain medulloblastoma.

a large set of constraints linking the information scattered in various knowledge sources. We have presented an efficient new algorithm MUSIC-DFS which soundly and completely mines such constrained patterns. Effectiveness comes from an interval pruning strategy based on prefix free patterns. To the best of our knowledge, there is no other contraint-based tool able to solve such constraint-based tasks.

The genomic case study demonstrates that our approach can handle large datasets. It also shows practical utility of the flexible framework integrating heterogeneous knowledge sources. The language of primitives applied to a wide spectrum of genomic data results in constraints formalizing viable notion of interestingness.

Acknowledgements. The authors thank the CGMC Laboratory (CNRS UMR 5534, Lyon) for providing the gene expression database and many valuable comments. This work has been partially funded by the ACI "masse de données" (French Ministry of research), Bingo project (MD 46, 2004-07).

References

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, pages 432–444, 1994.

[2] J. F. Boulicaut, A. Bykowski, and C. 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.

[3] C. Bucila, J. Gehrke, D. Kifer, and W. M. White. Dualminer: A dual-pruning algorithm for itemsets with constraints. *Data Min. Knowl. Discov.*, 7(3):241–272, 2003.

[4] C. Hébert and B. Crémilleux. Mining frequent δ-free patterns in large databases. In A. Hoffmann, H. Motoda, and T. Scheffer, editors, *proceedings of the 8th International Conference on Discovery Science (DS'05)*, volume 3735 of *Lecture notes in artificial intelligence*, pages 124–136, Singapore, 2005. Springer-Verlag.

[5] B. Jeudy and F. Rioult. Database transposition for constrained (closed) pattern mining. In *KDID*, volume 3377 of *Lecture Notes in Computer Science*, pages 89–107. Springer, 2004.

[6] J. Kléma, A. Soulet, B. Crémilleux, S. Blachon, and O. Gandrillon. Mining plausible patterns from genomic data. In D. Lee, B. Nutter, S. Antani, S. Mitra, and J. Archibald, editors, *CBMS 2006, the 19th IEEE International Symposium on Computer-Based Medical Systems*, pages 183–188, Salt Lake City, Utah, 2006.

[7] H. Mannila and H. Toivonen. Levelwise search and borders of theories in knowledge discovery. *Data Mining and Knowledge Discovery*, 1(3):241–258, 1997.

[8] F. Pan, G. Cong, A. K. H. Tung, Y. Yang, and M. J. Zaki. CARPENTER: finding closed patterns in long biological datasets. In *proceedings of the 9th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD'03)*, pages 637–642, Washington, DC, USA, 2003. ACM Press.

[9] N. Pasquier, Y. Bastide, T. Taouil, and L. Lakhal. Discovering frequent closed itemsets for association rules. *Lecture Notes in Computer Science*, 1540:398–416, 1999.

[10] J. Pei, J. Han, and L. V. S. Lakshmanan. Mining frequent item sets with convertible constraints. In *ICDE*, pages 433–442. IEEE Computer Society, 2001.

[11] A. Soulet and B. Crémilleux. An efficient framework for mining flexible constraints. In *PAKDD*, volume 3518 of *Lecture Notes in Computer Science*, pages 661–671. Springer, 2005.

[12] V. Velculescu, L. Zhang, B. Vogelstein, and K. Kinzler. Serial analysis of gene expression. *Science*, 270:484–7, 1995.

Author Index