A Lossless Data Reduction for Mining Constrained Patterns in n-ary Relations

Gabriel Poesia and Loïc Cerf

Department of Computer Science, Universidade Federal de Minas Gerais
Belo Horizonte, Brazil
{gabriel.poesia,lcerf}@dcc.ufmg.br

Abstract. Given a binary relation, listing the itemsets takes exponential time. The problem grows worse when searching for analog patterns defined in n-ary relations. However, real-life relations are sparse and, with a greater number n of dimensions, they tend to be even sparser. Moreover, not all itemsets are searched. Only those satisfying some user-defined constraints, such as minimal size constraints. This article proposes to exploit together the sparsity of the relation and the presence of constraints satisfying a common property, the monotonicity w.r.t. one dimension. It details a pre-processing step to identify and erase n-tuples whose removal does not change the collection of patterns to be discovered. That reduction of the relation is achieved in a time and a space that is linear in the number of n-tuples. Experiments on two real-life datasets show that, whatever the algorithm used afterward to actually list the patterns, the pre-process allows to lower the overall running time by a factor typically ranging from 10 to 100.

1 Introduction

Given a binary relation, which generically represents objects having (or not) some Boolean properties, an *itemset* is a subset of the properties. It can be associated with the subset of all objects having all those properties. Those objects are called the *support* of the itemset. Mining the itemsets with their supports allow the discovery of correlations between arbitrary numbers of Boolean properties, between arbitrary number of objects and between the objects and the properties. For instance, mining a binary relation indicating whether a customer (an object) bought a food item (a property) can unveil interesting buying behaviors. The pattern ({`Alice`,`Bob`,`Dave`}, {`bread`,`cheese`,`oil`,`salt`}) indicates that the three customers in the support are the only ones who bought together the four food items in the itemset. The number of itemsets is exponential in the number of properties and so is the time to compute them.

To keep under control the size of that output, two techniques are classically used. First of all, the itemsets that are not *closed* can be removed from the output without any loss of information. Every non-closed itemset is, by definition, strictly included into another itemset with the exact same support. For instance, {`bread`,`cheese`,`oil`,`salt`} is not closed if Alice, Bob and Dave

T. Calders et al. (Eds.): ECML PKDD 2014, Part II, LNCS 8725, pp. 581–596, 2014.

all bought some butter. If there is no other food item that they all bought, {`bread`,`cheese`,`oil`,`salt`,`butter`} is closed. With the support stored along with the closed itemset, the support of *any* itemset can easily be retrieved from the reduced collection of patterns: it is the support of the smallest superset that is closed. However the number of closed itemsets remains exponential in the number of objects or in the number of properties (whichever is smaller).

To further reduce the output, the sole *relevant* (closed) itemsets must be shown. The relevance is usually defined by the analyst as a conjunction of constraints that every output itemset must satisfy. For instance, knowing the subset W of customers who are women and the function p returning the price of the food item in argument, our analyst may want to take a look at the patterns (C, I) satisfying the following constraints:

$\mathcal{C}_{\geq 8\ \textbf{women}}(C, I) \equiv |C \cap W| \geq 8$ for at least eight women in every pattern;
$\mathcal{C}_{\leq 12\ \textbf{items}}(C, I) \equiv |I| \leq 12$ for at most twelve items in every pattern;
$\mathcal{C}_{50\textbf{-min-area}}(C, I) \equiv |C \times I| \geq 50$ for at least fifty tuples in the cover of every pattern;
$\mathcal{C}_{8\$\textbf{-max-price}}(C, I) \equiv \max_{i \in I} p(i) \leq 8$ for all items in every pattern having a price below 8$;
$\mathcal{C}_{4\$\textbf{-min-range-price}}(C, I) \equiv \max_{(i,i') \in I^2}(p(i) - p(i')) \geq 4$ for a price difference of at least 4$ between the cheapest and the most expensive item in every pattern;
$\mathcal{C}_{10\$\textbf{-min-total-price}}(C, I) \equiv \sum_{i \in I} p(i) \geq 10$ for at least 10$ worth of items in every pattern.

Depending on the algorithm at work, some constraints can guide the search of the itemsets, i. e., regions of the pattern space are left unexplored because they do not contain any relevant itemset. Doing so, the relevant patterns can be discovered in a fraction of the time required to list every unconstrained pattern.

When dealing with "big data", whose growth in quantity is steeper than that of disk sizes, the first technique that is commonly applied is to simply identify irrelevant data that need not be stored. Constraints on itemsets play this role on the "big data output". But what about using the constraints before the actual extraction to reduce the input data? The binary relation is not "big". Nevertheless, because listing the constrained itemsets generally remains NP-hard, that simple idea can lead to a great reduction of the overall running time. This is especially true when a constraint allows, in a pre-processing step, to remove some tuples but cannot be used by the chosen algorithm to prune the pattern space (hence the need for a filter at the output). Notice that the removed tuples must be guaranteely useless, i. e., with or without them, the closed itemsets satisfying the constraints must be the same.

Such a pre-processing method has already been proposed for (not necessarily closed) itemset mining [3]. In this article, the closedness is taken into consideration. More challenging, the task is generalized toward n-ary relations. For example, our proposal can take advantage of some of the constraints listed above in the context of a ternary relations that encode whether customers buy items along time (e. g., the third element of a 3-tuple can be `jan-14` or `feb-14` or etc.).

That pre-process works at the level of *tubes*, i. e., one dimensional subspaces of the n-ary relation such as ({`Alice`}, {`bread`},all months in which `Alice` bought `bread`), ({`Alice`},all items `Alice` bought in `jan-14`, {`jan-14`}) and (all customers buying `bread` in `jan-14`, {`bread`}, {`jan-14`}). The Cartesian product of the n dimensions of a tube is called the *cover* of this tube. The less n-tuples in the cover of a tube, the more likely they can be seamlessly removed altogether from the relation thanks to a constraint that is *monotone w.r.t one dimension*, a property that this article introduces and that many common constraints happen to satisfy. Moreover, every n-tuple is in the cover of n tubes "oriented" in each of the n dimensions of the relation. As a consequence, emptying a tube makes it more likely that some of the orthogonal tubes can be emptied in a sequence. Indeed, their covers have just lost one n-tuple.

The pre-process therefore is effective as long as the relation contains tubes with small covers. It turns out that real-life n-ary relations often are sparse and even sparser for a greater n. In our example, a customer who ever bought an item usually did not buy it every month and the ternary relation is sparser than the binary one. Furthermore, the distribution, over all tubes, of the number of covered tuples often is skewed, i. e., most of the tubes cover few n-tuples. All those n-tuples, covered by the long tail of the distribution, are prone to be removed by the pre-process. Figure 1 shows such a distribution for one of the real-life ternary relations we used in our experiments. Each curve relates to one "orientation" for the tubes. In the log-scaled abscissa, those tubes were ordered in decreasing order of the number of 3-tuples they cover.

After presenting the related work in Sect. 2, Sect. 3 provides some definitions and formally defines the data-mining problem we consider. In Sect. 4, the pre-process is detailed and its correctness proved. Sect. 5 shows, on two real-life datasets that it frequently allows to solve the problem orders of magnitude faster. Finally, Sect. 6 briefly concludes.

2 Related Work

Given a binary relation, which can be seen as a Boolean matrix, the famous Apriori algorithm [1] mines itemsets under a minimal frequency constraint, i. e., a minimal number of rows in the support of the itemset. Apriori first considers the individual columns of the matrix and removes those with a number of present tuples that is below the frequency threshold. Indeed, such columns cannot be involved in any frequent itemset. This property of the frequency constraint has later been called *anti-monotonicity* by opposition to *monotonicity* [9].

[5] and [6] are among the early studies of the efficient extraction of patterns under both monotone and anti-monotone constraints. A monotone constraint on the rows of a pattern is anti-monotone when applied, instead, on its columns. This duality obviously vanishes when considering Boolean tensors, i. e., relations of higher arities. In this article, the expression *monotonicity w.r.t. one dimension* is coined. In the specific context of a binary relation, a constraint is monotone w.r.t. rows (respectively columns) if it only deals with rows

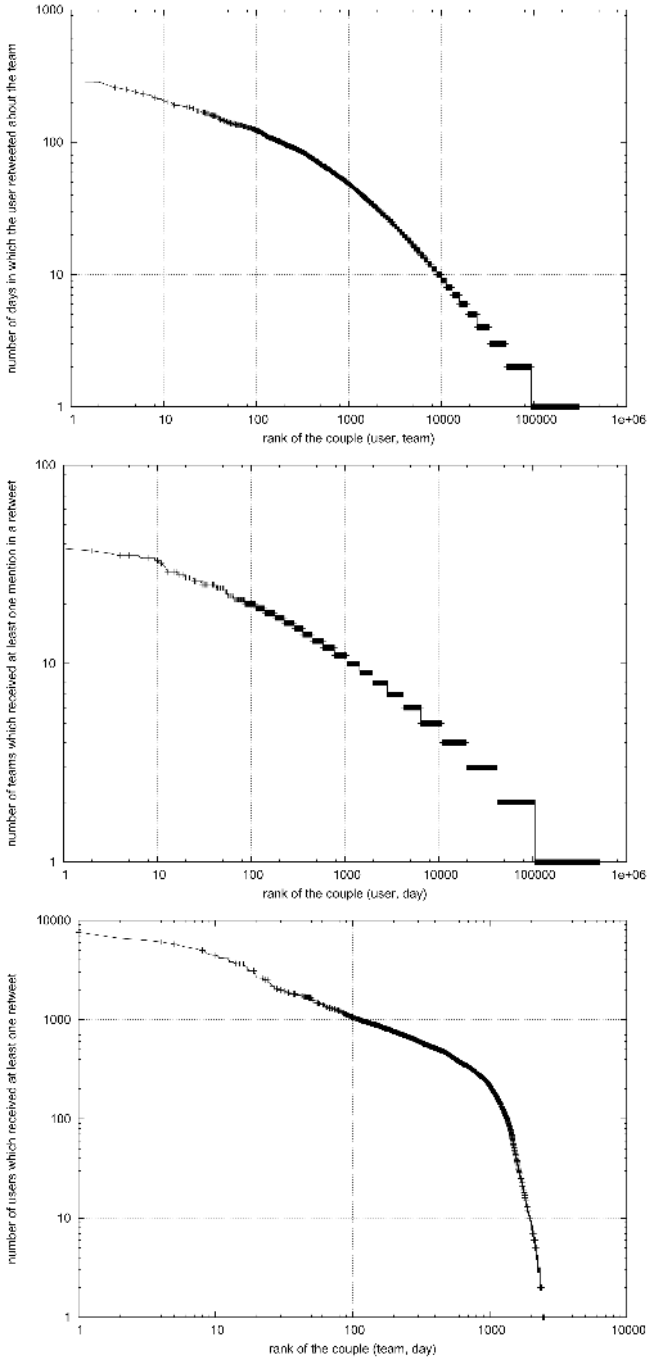

Fig. 1. Distributions of the number of 3-tuples in the cover of the tubes w.r.t. each of the three dimensions of the densest `Retweet` relation

(respectively columns) and is anti-monotone (respectively monotone) in the classical meaning of the word.

When constrained patterns are to be extracted from a binary relation, the individual rows (respectively columns) that do not satisfy the monotone (respectively anti-monotone) constraints can be removed. ExAnte [3] iteratively performs that reduction of a binary relation until a fixed point is reached, i. e., until no more row or column is removed. The algorithm presented in this article can be seen as a generalization of ExAnte toward n-ary relations ($n \geq 2$). To the best of our knowledge, it is the first attempt to pre-process an n-ary relation to speed up the subsequent search of constrained patterns.

Those "patterns" naturally generalize itemsets (along with their supports). More precisely, a pattern in an n-ary relation consists of n subsets of each of the n dimensions and only covers tuples present in the relation, i. e., the Cartesian product of the n subsets must be included in the relation. [11], [10] and [14] detail algorithms to mine such patterns in ternary relations. [7] and [13] directly tackle the search of patterns in arbitrary n-ary relations. All those algorithms actually enforce an additional maximality property, the closedness constraint, which is known to losslessly reduce the collection of patterns to the most informative ones [8]. Besides, they all are able to focus the search of the patterns on those having user-specified minimal numbers of elements in each of the dimensions. In fact, all of them but Data-Peeler [7] *only* consider minimal size constraints. In contrast, Data-Peeler can prune the search of the patterns with any *piecewise (anti)-monotone* constraint and, as a consequence, with any constraint that is *monotone w.r.t. one dimension* (as defined in this article).

3 Definitions and Problem Statement

All along the article, $\times$ denotes the Cartesian product and $\prod$ is used for the Cartesian product of an arbitrary number of sets. Given $n \in \mathbb{N}$ dimensions of analysis (i. e., n finite sets) $(D_i)_{i=1..n}$, the dataset is a relation $\mathcal{R} \subseteq \prod_{i=1}^{n} D_i$, i. e., a set of n-tuples. Table 1 represents such a relation $\mathcal{R}_E \subseteq \{\alpha, \beta, \gamma\} \times \{1, 2, 3, 4\} \times \{A, B, C\}$, hence a ternary relation. In this table, every '1' (resp. '0') at the intersection of three elements stands for the presence (resp. absence) of the related 3-tuple in $\mathcal{R}_E$. E. g., $(\alpha, 1, A) \in \mathcal{R}_E$ and $(\alpha, 1, C) \notin \mathcal{R}_E$.

Table 1. $\mathcal{R}_E \subseteq \{\alpha, \beta, \gamma\} \times \{1, 2, 3, 4\} \times \{A, B, C\}$

	A	B	C	A	B	C	A	B	C
1	1	1	0	0	0	1	0	1	0
2	0	0	0	1	1	0	1	1	0
3	1	0	1	0	1	1	1	0	0
4	1	0	0	1	0	0	0	1	0
		α			β			γ	

An *n-set* $(X_1, \cdots, X_n)$ consists of n subsets of each of the n dimensions, i. e., $(X_1, \cdots, X_n) \in \prod_{i=1}^{n} \mathcal{P}(D_i)$. For example, given the dimensions of $\mathcal{R}_E$, $(\{\alpha, \gamma\}, \{2, 4\}, \{B\})$ is an n-set, whereas $(\{\beta\}, \{4\}, \{A, \alpha\})$ is not because $\alpha \notin D_3$.

An *i-tube* (with $i \in \{1, \cdots, n\}$) is a special n-set: all its dimensions but the i^{th} are singletons and its i^{th} dimension contains all the elements in D_i that form, with the elements in the singletons, n-tuples present in the relation. Formally, $(T_1, \cdots, T_n) \in \prod_{i=1}^{n} \mathcal{P}(D_i)$ is an *i-tube* in $\mathcal{R} \subseteq \prod_{i=1}^{n} D_i$ if and only if:

$$\begin{cases} \forall j \in \{1, \cdots, i-1, i+1, \cdots, n\},\ \exists t_j \in D_j \mid T_j = \{t_j\} \\ T_i = \{t_i \in D_i \mid (t_1, \cdots, t_{i-1}, t_i, t_{i+1}, \cdots, t_n) \in \mathcal{R}\} \end{cases}.$$

$(\{\alpha\}, \{1, 3, 4\}, \{A\})$ is an example of a 2-tube in $\mathcal{R}_E$. $(\{\alpha\}, \{1, 3\}, \{A\})$ is not a 2-tube in $\mathcal{R}_E$ because 4 is not in its second dimension although $(\alpha, 4, A) \in \mathcal{R}_E$. $(\{\alpha\}, \{1, 2, 3, 4\}, \{A\})$ is not a 2-tube either because $(\alpha, 2, A) \notin \mathcal{R}_E$.

A *constraint* over an n-set is a predicate, i. e., a function that associates every n-set with a value of either true or false. The data mining task we consider is the extraction, from an n-ary relation, of *all* patterns ("pattern" will be defined in the next paragraph) satisfying a conjunction of constraints that are independent from the relation. The introduction of this article provides six examples of constraints. None of them depends on the relation, i. e., the sole n-set and, possibly, some external data (such as W and p in the introduction) are sufficient to evaluate the constraint.

A *pattern* in an n-ary relation is a natural generalization of a closed itemset and its support in a binary relation. It is an n-set (1) whose *cover* (the Cartesian product of its n dimensions) is included in the relation and (2) that is closed. The closedness is a property of maximality. It means that no element can be added to any of dimension of the pattern without breaking property (1). Formally, $(X_1, \cdots, X_n) \in \prod_{i=1}^{n} \mathcal{P}(D_i)$ is a pattern in $\mathcal{R} \subseteq \prod_{i=1}^{n} D_i$ if and only if:

$$\begin{cases} (1)\ \prod_{i=1}^{n} X_i \subseteq \mathcal{R} \\ (2)\ \forall (X'_1, \cdots, X'_n) \in \prod_{i=1}^{n} \mathcal{P}(D_i), \\ \quad \begin{cases} \forall i \in \{1, \cdots, n\}, X_i \subseteq X'_i \\ \prod_{i=1}^{n} X'_i \subseteq \mathcal{R} \end{cases} \Rightarrow \forall i \in \{1, \cdots, n\}, X_i = X'_i \end{cases}.$$

For brevity, we sometimes write $\mathcal{C}_{\text{closed}}(X_1, \cdots, X_n, \mathcal{R})$ to mean that the n-set $(X_1, \cdots, X_n)$ satisfies property (2).

$(\{\beta, \gamma\}, \{2\}, \{A, B\})$ is an example of a pattern in $\mathcal{R}_E$ because (1) the four 3-tuples in $\{\beta, \gamma\} \times \{2\} \times \{A, B\}$ belong to $\mathcal{R}_E$ and (2) no element can be added to any of its dimensions without breaking property (1). $(\{\beta\}, \{2\}, \{A, B\})$ is not a pattern in $\mathcal{R}_E$ because it is not closed: it is "included" in $(\{\beta, \gamma\}, \{2\}, \{A, B\})$, which only covers 3-tuples that are present in $\mathcal{R}_E$. $(\{\alpha, \gamma\}, \{1\}, \{A\})$ is not a pattern either because $(\gamma, 1, A)$ can be formed by taking an element in each of its dimensions and $(\gamma, 1, A) \notin \mathcal{R}_E$.

The data mining task considered in this paper can now be formalized. Given a relation $\mathcal{R} \subseteq \prod_{i=1}^{n} D_i$ and a set $\mathcal{C}_{\text{all}}$ of constraints that are all independent from the relation, the problem is the computation of the following set $\mathcal{T}h(\mathcal{R}, \mathcal{C}_{\text{all}})$:

$$\left\{(X_1,\cdots,X_n)\in\prod_{i=1}^{n}\mathcal{P}(D_i)\mid\begin{cases}(X_1,\cdots,X_n)\text{ is a pattern in }\mathcal{R}\\\forall\mathcal{C}\in\mathcal{C}_{\text{all}},\mathcal{C}(X_1,\cdots,X_n)\end{cases}\right\}$$

However, this work is not about a new algorithm to solve the problem. It is about computing a relation $\mathcal{R}'\subseteq\mathcal{R}$ that is as small as possible and yet guarantees that $\mathcal{T}h(\mathcal{R}',\mathcal{C}_{\text{all}})=\mathcal{T}h(\mathcal{R},\mathcal{C}_{\text{all}})$. In this way, the actual pattern miner potentially runs faster on $\mathcal{R}'$ and yet outputs the correct and complete collection of constrained patterns.

To shrink $\mathcal{R}$ into $\mathcal{R}'$, the algorithm proposed in this article exploits the constraints in $\mathcal{C}_{\text{all}}$ that are *monotone w.r.t. one dimension*. They are constraints that do not depend on any dimension of the n-set but one and that are monotone w.r.t. the inclusion order on this dimension, i. e., if an n-set satisfies a constraint that is monotone w.r.t. dimension i, then any n-set with a larger i^{th} dimension (w.r.t. set inclusion) satisfies it as well. Formally, a constraint $\mathcal{C}$ is *monotone w.r.t. dimension* i (with $i\in\{1,\cdots,n\}$) if and only if:

$$\begin{gathered}\forall(X_1,\cdots,X_n)\in\textstyle\prod_{i=1}^{n}\mathcal{P}(D_i),\forall Y_i\subseteq D_i,\\\mathcal{C}(X_1,\cdots,X_n)\Rightarrow\mathcal{C}(X_1,\cdots,X_{i-1},X_i\cup Y_i,X_{i+1},\cdots,X_n)\ .\end{gathered}$$

Among the six constraints listed in the introduction, $\mathcal{C}_{\geq 8\text{ women}}$ is monotone w.r.t. the customer dimension; $\mathcal{C}_{8\$\text{-max-price}}$, $\mathcal{C}_{4\$\text{-min-range-price}}$ and $\mathcal{C}_{10\$\text{-min-total-price}}$ are monotone w.r.t. the food item dimension. $\mathcal{C}_{\leq 12\text{ items}}$ is not monotone: given an n-set that satisfies it, there exists another n-set with more than twelve items, including all those involved in the first n-set. $\mathcal{C}_{50\text{-min-area}}$ is not monotone w.r.t. one dimension either because it depends on two dimensions of the n-set.

4 Dataset Reduction

The reduction of the n-ary relation $\mathcal{R}\subseteq\prod_{i=1}^{n}D_i$, which is proposed in this article, is based on the removal of the n-tuples covered by an i-tube that does not verify a constraint that is monotone w.r.t. dimension i. This section first proves that this operation does not change the set of constrained patterns that is to be discovered. Then the actual algorithm is presented and its complexity is analyzed.

4.1 Fundamental Theorem

Let us first show that the i^{th} dimension of a pattern necessarily contains a subset of the i^{th} dimension of any i-tube that covers some of its tuples:

Lemma 1. *Given a pattern $(X_1,\cdots,X_n)$ in $\mathcal{R}$ and an i-tube $(T_1,\cdots,T_n)$ in $\mathcal{R}$ (with $i\in\{1,\cdots,n\}$), we have:*

$$\left(\prod_{j=1}^{n}X_j\right)\cap\left(\prod_{j=1}^{n}T_j\right)\neq\varnothing\Rightarrow X_i\subseteq T_i\ .$$

Proof. If $\left(\prod_{j=1}^{n} X_j\right) \cap \left(\prod_{j=1}^{n} T_j\right) \neq \varnothing$, then $\forall j \in \{1, \cdots, n\}$, $X_j \cap T_j \neq \varnothing$. By definition of the i-tube $(T_1, \cdots, T_n)$, $\forall j \neq i$, $|T_j| = 1$. As a consequence, $\forall j \neq i$, $T_j \subseteq X_j$ (1). Assume, by contradiction, $X_i \not\subseteq T_i$, i. e., $\exists e \in X_i \setminus T_i$. By (1) and the first property defining the pattern $(X_1, \cdots, X_n)$, $T_1 \times \cdots T_{i-1} \times \{e\} \times T_{i+1} \times \cdots \times T_n \subseteq \prod_{j=1}^{n} X_j \subseteq \mathcal{R}$ (2). By definition of the i-tube $(T_1, \cdots, T_n)$, $T_i = \{t_i \in D_i \mid (t_1, \cdots, t_{i-1}, t_i, t_{i+1}, \cdots, t_n) \in \mathcal{R}\}$. With (2), we therefore have $e \in T_i$, which contradicts the assumption. □

From now on, $\mathcal{C}_i$ (with $i \in \{1, \cdots, n\}$) denotes the conjunction of all constraints in $\mathcal{C}_{\text{all}}$ that are monotone w.r.t. dimension i. Clearly, $\mathcal{C}_i$ is monotone w.r.t. dimension i. The following lemma states that whenever an i-tube violates $\mathcal{C}_i$, removing the n-tuples covered by this i-tube leads to a reduced relation that does not embed any pattern absent from the original relation.

Lemma 2. *Given $\mathcal{R}$, $\mathcal{C}_{all}$ and an i-tube $(T_1, \cdots, T_n)$ in $\mathcal{R}$ (with $i \in \{1, \cdots, n\}$), we have:*

$$\neg\mathcal{C}_i(T_1, \cdots, T_n) \Rightarrow \mathcal{T}h\left(\mathcal{R} \setminus \prod_{j=1}^{n} T_j, \mathcal{C}_{all}\right) \subseteq \mathcal{T}h\left(\mathcal{R}, \mathcal{C}_{all}\right) \quad .$$

Proof. Let $(X_1, \cdots, X_n) \in \mathcal{T}h(\mathcal{R} \setminus \prod_{j=1}^{n} T_j, \mathcal{C}_{\text{all}})$.

By the first property defining the pattern $(X_1, \cdots, X_n)$ in $\mathcal{R} \setminus \prod_{j=1}^{n} T_j$, $\prod_{j=1}^{n} X_j \subseteq \mathcal{R} \setminus \prod_{j=1}^{n} T_j$. Because $\mathcal{R} \setminus \prod_{j=1}^{n} T_j \subseteq \mathcal{R}$ and by transitivity of $\subseteq$, $\prod_{j=1}^{n} X_j \subseteq \mathcal{R}$ (1).

Assume, by contradiction, $\neg\mathcal{C}_{\text{closed}}(X_1, \cdots, X_n, \mathcal{R})$. By definition of $\mathcal{C}_{\text{closed}}$, $\exists (X'_1, \cdots, X'_n) \in \prod_{i=1}^{n} \mathcal{P}(D_i) \mid \begin{cases} \prod_{i=1}^{n} X'_i \subseteq \mathcal{R}(2) \\ \forall i \in \{1, \cdots, n\}, X_i \subseteq X'_i(3) \\ \exists i \in \{1, \cdots, n\} \mid X_i \subsetneq X'_i(4) \end{cases}$. We necessarily have $\prod_{i=1}^{n} X'_i \not\subseteq (\mathcal{R} \setminus \prod_{i=1}^{n} T_i)$ otherwise, with (3) and (4), it would follow that $\neg C_{\text{closed}}(X_1, \cdots, X_n, \mathcal{R} \setminus \prod_{j=1}^{n} T_j)$ what would contradict $(X_1, \cdots, X_n) \in \mathcal{T}h(\mathcal{R} \setminus \prod_{j=1}^{n} T_j, \mathcal{C}_{\text{all}})$. By adding (2) to that, we have $(\prod_{j=1}^{n} X'_j) \cap (\prod_{j=1}^{n} T_j) \neq \varnothing$ and, by Lemma 1, $X'_i \subseteq T_i$. Because (3) imposes $X_i \subseteq X'_i$, we have, by transitivity of $\subseteq$, $X_i \subseteq T_i$. Therefore, by contraposition of the definition of the monotonicity w.r.t. dimension i that holds for $\mathcal{C}_i$, $\neg\mathcal{C}_i(T_1, \cdots, T_n) \Rightarrow \neg\mathcal{C}_i(X_1, \cdots, X_n)$. As a consequence, $(X_1, \cdots, X_n) \notin \mathcal{T}h(\mathcal{R} \setminus \prod_{j=1}^{n} T_j, \mathcal{C}_{\text{all}})$, a contradiction. Therefore, the assumption is wrong, i. e., $\mathcal{C}_{\text{closed}}(X_1, \cdots, X_n, \mathcal{R})$ (5).

Finally, because all constraints in $\mathcal{C}_{\text{all}}$ are independent from the relation, the fact that $(X_1, \cdots, X_n)$ satisfies them in $\mathcal{R} \setminus \prod_{j=1}^{n} T_j$ implies that it satisfies them as well in $\mathcal{R}$. Together with (1) and (5), we therefore have $(X_1, \cdots, X_n) \in \mathcal{T}h(\mathcal{R}, \mathcal{C}_{\text{all}})$. □

One final lemma to state the opposite of Lemma 2, i. e., whenever an i-tube violates $\mathcal{C}_i$, removing the n-tuples covered by this i-tube leads to a reduced relation that embeds every pattern present in the original relation.

Lemma 3. *Given $\mathcal{R}$, $\mathcal{C}_{all}$ and an i-tube $(T_1, \cdots, T_n)$ in $\mathcal{R}$ (with $i \in \{1, \cdots, n\}$), we have:*

$$\neg\mathcal{C}_i(T_1, \cdots, T_n) \Rightarrow \mathcal{T}h\left(\mathcal{R}, \mathcal{C}_{all}\right) \subseteq \mathcal{T}h\left(\mathcal{R} \setminus \prod_{j=1}^{n} T_j, \mathcal{C}_{all}\right) .$$

Proof. Let $(X_1, \cdots, X_n) \in \mathcal{T}h(\mathcal{R}, \mathcal{C}_{\text{all}})$.

By the first property defining the pattern $(X_1, \cdots, X_n)$ in $\mathcal{R}$, $\prod_{j=1}^{n} X_j \subseteq \mathcal{R}$ (1). Assume, by contradiction, $\prod_{j=1}^{n} X_j \not\subseteq \mathcal{R} \setminus \prod_{j=1}^{n} T_j$. With (1), we have $(\prod_{j=1}^{n} X_j) \cap (\prod_{j=1}^{n} T_j) \neq \varnothing$, i. e., Lemma 1 applies and $X_i \subseteq T_i$. By contraposition of the definition of the monotonicity w.r.t. dimension i that holds for $\mathcal{C}_i$, $\neg\mathcal{C}_i(T_1, \cdots, T_n) \Rightarrow \neg\mathcal{C}_i(X_1, \cdots, X_n)$. As a consequence, $(X_1, \cdots, X_n) \notin \mathcal{T}h(\mathcal{R}, \mathcal{C}_{\text{all}})$, a contradiction. Therefore, the assumption is wrong, i. e., $\prod_{j=1}^{n} X_j \subseteq \mathcal{R} \setminus \prod_{j=1}^{n} T_j$ (2).

$\mathcal{C}_{\text{closed}}(X_1, \cdots, X_n, \mathcal{R} \setminus \prod_{j=1}^{n} T_j)$ (3) directly follows from $\mathcal{R} \setminus \prod_{j=1}^{n} T_j \subseteq \mathcal{R}$ and, in a sequence, from the closedness of $(X_1, \cdots, X_n)$ in $\mathcal{R}$: $\forall (X'_1, \cdots, X'_n) \in \prod_{j=1}^{n} \mathcal{P}(D_j)$, $\begin{cases} \forall j \in \{1, \cdots, n\}, X_j \subseteq X'_j \\ \prod_{j=1}^{n} X'_j \subseteq \mathcal{R} \setminus \prod_{j=1}^{n} T_j \subseteq \mathcal{R} \end{cases} \Rightarrow \forall j \in \{1, \cdots, n\}, X_j = X'_j.$

Finally, because all constraints in $\mathcal{C}_{\text{all}}$ are independent from the relation, the fact that $(X_1, \cdots, X_n)$ satisfies them in $\mathcal{R}$ implies that it satisfies them as well in $\mathcal{R} \setminus \prod_{j=1}^{n} T_j$. Together with (2) and (3), we therefore have $(X_1, \cdots, X_n) \in \mathcal{T}h(\mathcal{R} \setminus \prod_{j=1}^{n} T_j, \mathcal{C}_{\text{all}})$. □

Finally, here is the theorem at the foundation of the data reduction proposed in this article.

Theorem 1. *Given $\mathcal{R}$, $\mathcal{C}_{all}$ and an i-tube $(T_1, \cdots, T_n)$ in $\mathcal{R}$ (with $i \in \{1, \cdots, n\}$), we have:*

$$\neg\mathcal{C}_i(T_1, \cdots, T_n) \Rightarrow \mathcal{T}h\left(\mathcal{R}, \mathcal{C}_{all}\right) = \mathcal{T}h\left(\mathcal{R} \setminus \prod_{j=1}^{n} T_j, \mathcal{C}_{all}\right) .$$

Proof. The equality follows from Lemmas 2 and 3.

4.2 Algorithm

The obvious pre-process, which directly follows from Th. 1, would consider, one by one and for all $i \in \{1, \cdots, n\}$, every i-tube in $\mathcal{R}$. It would test whether the related $\mathcal{C}_i$ is satisfied and, if not, it would "empty" the i-tube. However, the removal of an n-tuple in an i-tube corresponds as well to the removal of this same n-tuple in every orthogonal j-tube (with $j \neq i$). Such a j-tube may have already been considered and was satisfying $\mathcal{C}_j$. However, since $\mathcal{C}_j$ is monotone w.r.t. dimension j, the j-tube may now violate $\mathcal{C}_j$ because it contains one element less in its j^{th} dimension. In this way, the constraints that are monotone w.r.t. one dimension work in synergy with the constraints that are monotone w.r.t.

any other dimension. When one is effective, (i. e., allows to identify a tube to empty), it makes it more likely that the others become effective.

The following pseudo-code formalizes the pre-process. It enumerates, one by one, the n-tuples in the relation and checks all n tubes that cover each of the n-tuples. Whenever an i-tube is emptied because it violates $\mathcal{C}_i$, the j-tubes ($j \neq i$) that involve the removed n-tuples are rechecked. In this way, the pre-process only terminates when all i-tubes, for all $i \in \{1, \cdots, n\}$, are either empty or satisfy the related constraint $\mathcal{C}_i$.

Data: relation $\mathcal{R} \subseteq \prod_{j=1}^{n} D_j$, set $\mathcal{C}_{\text{all}}$ of constraints that are all independent from $\mathcal{R}$
begin
  forall the $(t_1, \cdots, t_n) \in \mathcal{R}$ **do**
    forall the $i \in \{1, \cdots, n\}$ **do**
      CleanTube($\mathcal{R}$, i, $(t_1, \cdots, t_{i-1}, t_{i+1}, \cdots, t_n)$);

Algorithm 1. CleanRelation

Data: relation $\mathcal{R} \subseteq \prod_{j=1}^{n} D_j$, orientation of the tube $i \in \{1, \cdots, n\}$, elements in the singletons of the tube $(t_1, \cdots, t_{i-1}, t_{i+1}, \cdots, t_n)$
begin
  $T_i \leftarrow \{t_i \in D_i \mid (t_1, \cdots, t_{i-1}, t_i, t_{i+1}, \cdots, t_n) \in \mathcal{R}\}$;
  if $\neg\mathcal{C}_i(\{t_1\}, \cdots, \{t_{i-1}\}, T_i, \{t_{i+1}\}, \cdots \{t_n\})$ **then**
    forall the $t_i \in T_i$ **do**
      $\mathcal{R} \leftarrow \mathcal{R} \setminus \{(t_1, \cdots, t_n)\}$;
      forall the $j \in \{1, \cdots, i-1, i+1, \cdots, n\}$ **do**
        CleanTube($\mathcal{R}$, j, $(t_1, \cdots, t_{j-1}, t_{j+1}, \cdots, t_n)$);

Procedure. CleanTube()

In the pseudo-code, there may be no removal of n-tuples in an i-tube between two checks of this i-tube. To avoid that, the actual implementation does not directly execute the recursive calls of CleanTube. Instead, the tubes in arguments of those calls are stored in a hash set (hence no duplicate). As long as the hash set is non-empty, a tube is retrieved from it and the related call of CleanTube is made. Once the hash map is empty, the execution comes back to CleanRelation.

Also, the tubes are not actually computed from the set of all n-tuples whenever they are required. Instead, the n-ary relation is stored n times as the set all i-tubes ($i \in \{1, \cdots, n\}$).

4.3 Complexity Analysis

In the worst case scenario, CleanRelation's enumeration of the n-tuples does not identify any tube to empty but the last one. Then, every single n-tuple is removed one by one, hence $|\mathcal{R}|$ calls of the CleanTube function. In this scenario, the pre-process consists of four steps whose time complexities follow:

Storage of $\mathcal{R}$: $O(n|\mathcal{R}|)$ since every n-tuple in $\mathcal{R}$ is stored n times; it is the space complexity of the overall pre-process too (assuming no external data is required to verify or speed up the verification of some constraints);

Outer-most enumeration: $O(|\mathcal{R}| \sum_{i=1}^{n} \text{check}(\mathcal{C}_i))$ where $\text{check}(\mathcal{C}_i)$ denotes the cost of verifying whether one i-tube verifies the constraints in $\mathcal{C}_{\text{all}}$ that are monotone w.r.t. dimension i;

Actual cleaning: $O(|\mathcal{R}| \sum_{i=1}^{n} \text{check}(\mathcal{C}_i))$;

Output of the remaining n-tuples: $O(|\mathcal{R}|)$; the worst-case scenario for this step corresponds to no actual cleaning.

Overall, the pre-process has a $O(n|\mathcal{R}|)$ space complexity and a time complexity of $O(|\mathcal{R}| \sum_{i=1}^{n} \text{check}(\mathcal{C}_i))$. Notice that, for common constraints, $\text{check}(\mathcal{C}_i)$ is cheap. For instance, it is $O(1)$ for minimal size constraints (assuming every i-tube is stored in a container with a constant time access to its size) or minimal sum constraints over positive numbers (assuming the sums for each i-tube are stored and updated whenever an n-tuple in it is erased). It is $O(\log |D_i|)$ for a maximal, a minimal or a min-range constraint (using respectively max-heaps, min-heaps and both).

5 Experimental Study

CleanRelation is integrated to Data-Peeler, which is free software[1]. It is implemented in C++ and compiled by GCC 4.7.2 with the O3 optimizations. Because it is a pre-process, any pattern extractor can work on the reduced relation it outputs. We received, from their respective authors, the implementations of CubeMiner [11], Trias [10], Data-Peeler [7], TriCons [14] and CnS-Miner [13], i. e., all (exact) pattern extractor that handle ternary relations (or more in the cases of Data-Peeler and CnS-Miner). Unfortunately, CnS-Miner never produced any output and we therefore decided to focus the experimental study on ternary relations where comparisons can be made. TriCons did not work, either crashing or returning an incomplete output.

The remaining three algorithms are tested on a GNU/Linux™ system running on top of 3.10GHz cores and 12GB of RAM. CubeMiner and Data-Peeler, both implemented in C++, were compiled with GCC 4.7.2. Trias was compiled and interpreted by Oracle's JVM version 1.7.0_45. CubeMiner and Trias can only prune the search space with minimal size constraints on some or all dimensions of the pattern. In contrast, Data-Peeler's traversal of the

[1] It is available, under the terms of the GNU GPLv3, at `http://dcc.ufmg.br/~lcerf/en/prototypes.html#d-peeler`.

pattern space can be guided by any number of piecewise (anti)-monotone constraints. That includes any constraint that is monotone w.r.t. one dimension. As a consequence, unless the sole minimal size constraints are desired, DATA-PEELER would be preferred. Fortunately, minimal size constraints are monotone w.r.t. one dimension. The remainder of this section compares the times CUBEMINER, TRIAS and DATA-PEELER take to list minimally sized patterns in ternary relations, with and without the pre-process

5.1 `Retweet` Dataset

The micro-blogging service Twitter is particularly popular in Brazil. Tweets about the Brazilian soccer championship were collected from January, 9$^{\text{th}}$ 2014 to April, 11$^{\text{th}}$ 2014 (92 days) and classified w.r.t. to the mentioned team(s) (supervised classification method, which is out of the scope of this paper). How many times a user is retweeted (i. e., other users "repeat" her tweets) is known to be a good measure of her influence [12]. 184,159 users were retweeted at least once during the considered period. A 3-dimensional tensor gives how many times each of them is retweeted (over all her messages) during a given day when writing about a given soccer team (among 29). That tensor contains 731,685 non-null values.

It is turned into a ternary relation by keeping the tuples relating to cells of the tensors with a high enough number of retweets. In the experiments on this dataset, the threshold is a variable. On the contrary, the minimal size constraints on the patterns are kept constant: at least two days, two teams and two users. Although those constraints are rather loose, the pre-process is efficient because the relation is very sparse. In the most challenging context, when one retweet is considered "influential enough", CLEANRELATION only keeps 263,413 out of the 731,685 3-tuples, a 64% reduction.

5.2 `Distrowatch` Dataset

DistroWatch[2] is a popular Web site that gathers comprehensive information about GNU/Linux$^{\text{TM}}$, BSD, and Solaris operating systems. Every distribution is described on a separate page. When a visitor loads a page, her country is known from the IP address. The logs of the Web server are turned into a ternary relation that gives for any time period (13 semesters from early 2004 to early 2010) and every page (describing 655 distributions), the countries that visited it more than 25 times. From that relation, we consider the extraction of all patterns involving at least four semesters, m distributions and m countries, where m is an integer variable ranging from 5 to 48.

The relation contains 150,834 3-tuples, a number that is comparable to those of the `Retweet` relations. However, it is considerably denser. Because of that, and even with strong minimal size constraints, some of the algorithms cannot mine the patterns in the relation that is not pre-processed. Those same algorithms

[2] `http://www.distrowatch.com`

benefit a lot from the pre-process, with overall running times that become several orders of magnitude shorter. With $m = 10$, CleanRelation keeps 65,130 out of the 150,834 3-tuples, a 57% reduction.

5.3 Pre-processing Time

Figure 2 depicts Data-Peeler's running times on the `Retweet` relations with and without the pre-processing step. The actual data reduction takes only a fragment of the time required by the subsequent extraction. Despite the loose constraints at work ("at least two elements in every dimension of the pattern"), the pre-process is effective. With it, Data-Peeler lists all the constrained patterns in about one percent of the time it takes to process the non-reduced relation (for the exact same result).

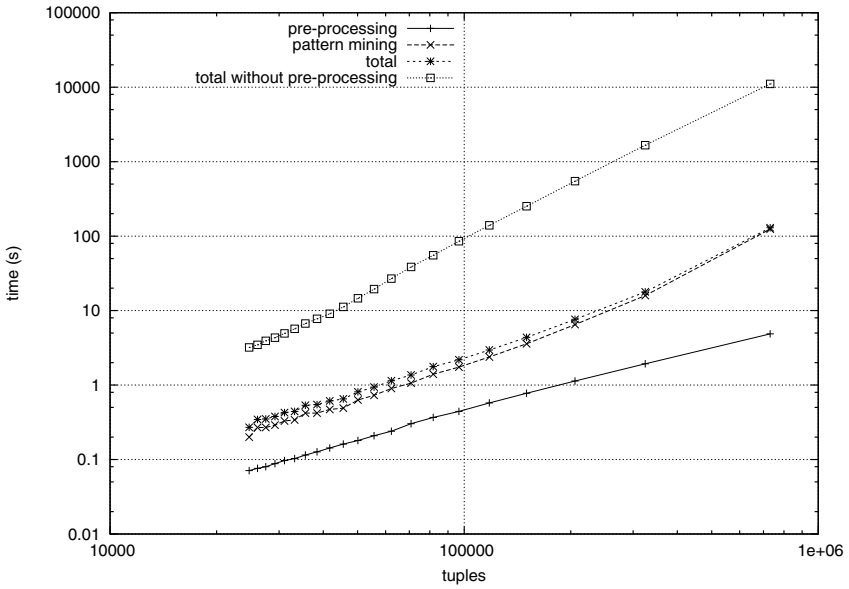

Fig. 2. Running times of CleanRelation and Data-Peeler running with and without the pre-processing step on the `Retweet` relations

5.4 Time Gains over the Whole Task

Figures 3 and 4 show the time gains CleanRelation brings to all the three tested algorithms. 24 hours are not enough for CubeMiner to directly mine the patterns in the `DistroWatch` dataset, even under the strongest considered constraints. However, in this same context but with the pre-process, it returns those patterns in 0.028s, i. e., at least three million times faster than without CleanRelation. Data-Peeler, which is the fastest algorithm when no pre-process is used, remains the fastest when it is used. However Data-Peeler

benefits less from CleanRelation than CubeMiner. The pre-process allows to divide the overall running time by a factor ranging between 2 and 4. Trias is 5 to 100 times faster when it mines the reduced relation rather than the original one. It is faster for Trias to compute from the reduced relation all patterns with at least four semesters, 16 distributions and 16 countries than to compute from the original relations the patterns with at least four semesters, 23 distributions and 23 countries.

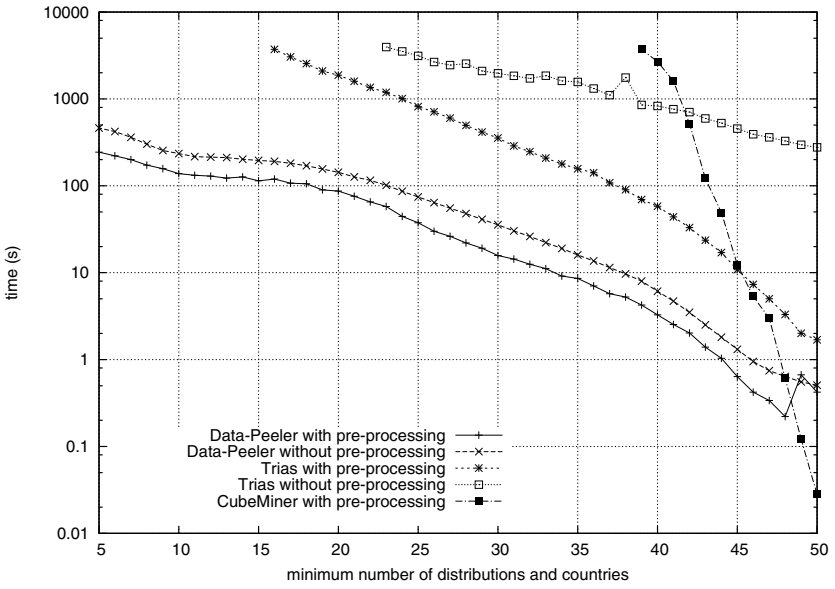

Fig. 3. Running times of Data-Peeler, CubeMiner and Trias with and without the pre-processing step on the `Distrowatch` relation

CubeMiner first computes an amount of memory to allocate for the pattern space. With an unreduced `Retweet` dataset, that amount overflows the number of bits in an integer and CubeMiner crashes. On the other hand, when mining the reduced relation, CubeMiner is efficient. It even competes with Data-Peeler for the sparsest versions of the dataset. Within a few hours, Trias manages to extract the constrained patterns only if the relation is very sparse. By preceding the call of Trias by the pre-process, the results dramatically improve. The running times are divided by about 50,000. Data-Peeler is the only algorithm that allows to extract, in a reasonable time, the patterns in the densest versions of the dataset. The pre-processing step helps it a lot in those more challenging contexts. In the dataset encoding whether a user was retweeted at least once when writing about a team during a day, the ternary relation, which used to contain 731,685 3-tuples, is reduced to only 170,388 tuples ($\approx$ 23% of the original size). In sequence, Data-Peeler runs about 100 times faster on the reduced relation.

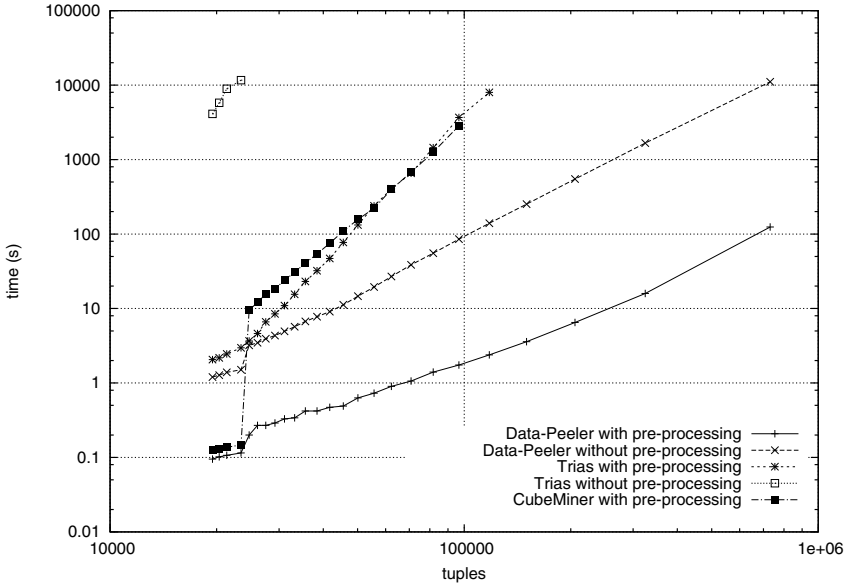

Fig. 4. Running times of Data-Peeler, CubeMiner and Trias with and without the pre-processing step on the `Retweet` relations

6 Conclusion

When searching for itemset-like patterns in an n-ary relation, constraints are, in practice, required. They specify some relevance criteria every pattern must satisfy, reduce the output to a manageable size and drastically lower the extraction times (if the algorithm can prune the search space with the constraints). In this article, we have identified a common property among constraints, the monotonicity w.r.t. one dimension, that allows to empty tubes (i. e., one-dimensional subspaces) of the relation while guaranteeing the presence of the same constrained patterns in the reduced data. Because an n-tuple belongs to n different tubes, constraints on the different dimensions of the pattern (e. g., minimal size constraints) work in synergy: emptying a tube makes it easier to empty the orthogonal tubes that used to contain the erased n-tuples. Once the fixed point reached, the actual pattern extraction, with any algorithm, takes place. Because real-life n-ary relations usually are sparse, the effectiveness of our pre-process can be impressive: in our experiments, the overall time to mine the patterns with the fastest algorithm, Data-Peeler, is lowered by a factor typically ranging from 10 to 100 and it can reach millions for less efficient algorithms. In the same way that the idea behind ExAnte [3] was then applied along the search for the itemsets [2,4], we currently investigate an analog integration of the present proposal into Data-Peeler.

References

1. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules in large databases. In: VLDB 1994: Proceedings of the 20th International Conference on Very Large Data Bases, pp. 487–499. Morgan Kaufmann (1994)
2. Bonchi, F., Giannotti, F., Mazzanti, A., Pedreschi, D.: ExAMiner: Optimized level-wise frequent pattern mining with monotone constraints. In: ICDM 2003: Proceedings of the 3rd International Conference on Data Mining, pp. 11–18. IEEE Computer Society (2003)
3. Bonchi, F., Giannotti, F., Mazzanti, A., Pedreschi, D.: ExAnte: Anticipated data reduction in constrained pattern mining. In: Lavrač, N., Gamberger, D., Todorovski, L., Blockeel, H. (eds.) PKDD 2003. LNCS (LNAI), vol. 2838, pp. 59–70. Springer, Heidelberg (2003)
4. Bonchi, F., Goethals, B.: FP-Bonsai: the art of growing and pruning small FP-trees. In: Dai, H., Srikant, R., Zhang, C. (eds.) PAKDD 2004. LNCS (LNAI), vol. 3056, pp. 155–160. Springer, Heidelberg (2004)
5. Boulicaut, J.F., Jeudy, B.: Using constraints during set mining: should we prune or not? In: BDA 2000: Actes des 16ème Journées Bases de Données Avancées, pp. 221–237 (2000)
6. Bucila, C., Gehrke, J., Kifer, D., White, W.M.: DualMiner: a dual-pruning algorithm for itemsets with constraints. In: KDD 2002: Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 42–51. ACM Press (2002)
7. Cerf, L., Besson, J., Robardet, C., Boulicaut, J.F.: Closed patterns meet n-ary relations. ACM Transactions on Knowledge Discovery from Data 3(1), 1–36 (2009)
8. Gallo, A., Mammone, A., Bie, T.D., Turchi, M., Cristianini, N.: From frequent itemsets to informative patterns. Tech. Rep. 123936, University of Bristol, Senate House, Tyndall Avenue, Bristol BS8 1TH, UK (December 2009)
9. Grahne, G., Lakshmanan, L.V.S., Wang, X.: Efficient mining of constrained correlated sets. In: ICDE 2000: Proceedings of the 16th International Conference on Data Engineering, pp. 512–521. IEEE Computer Society (2000)
10. Jaschke, R., Hotho, A., Schmitz, C., Ganter, B., Stumme, G.: Trias–an algorithm for mining iceberg tri-lattices. In: ICDM 2006: Proceedings of the 6th IEEE International Conference on Data Mining, pp. 907–911. IEEE Computer Society (2006)
11. Ji, L., Tan, K.L., Tung, A.K.H.: Mining frequent closed cubes in 3D data sets. In: VLDB'06: Proceedings of the 32nd International Conference on Very Large Data Bases, pp. 811–822. VLDB Endowment (2006)
12. Kwak, H., Lee, C., Park, H., Moon, S.: What is twitter, a social network or a news media? In: WWW 2010: Proceedings of the 19th International World Wide Web Conferences, pp. 591–600. ACM Press (2010)
13. Nataraj, R.V., Selvan, S.: Closed pattern mining from n-ary relations. International Journal of Computer Applications 1(9), 9–13 (2010)
14. Trabelsi, C., Jelassi, N., Ben Yahia, S.: Scalable mining of frequent tri-concepts from *Folksonomies*. In: Tan, P.-N., Chawla, S., Ho, C.K., Bailey, J. (eds.) PAKDD 2012, Part II. LNCS, vol. 7302, pp. 231–242. Springer, Heidelberg (2012)