Generalizing Itemset Mining in a Constraint Programming Setting

Jérémy Besson, Jean-François Boulicaut, Tias Guns, and Siegfried Nijssen

Abstract In recent years, a large number of algorithms have been proposed for finding set patterns in boolean data. This includes popular mining tasks based on, for instance, frequent (closed) itemsets. In this chapter, we develop a common framework in which these algorithms can be studied thanks to the principles of constraint programming. We show how such principles can be applied both in specialized and general solvers.

1 Introduction

Detecting local patterns has been studied extensively during the last decade (see, e.g., [18] and [22] for dedicated volumes). Among others, many researchers have considered the discovery of relevant set patterns (e.g., frequent itemsets and association rules, maximal itemsets, closed sets) from transac-

Jérémy Besson
Vilnius University, Faculty of Mathematics and Informatics
Naugarduko St. 24, LT-03225 Vilnius, Lithuania
e-mail: contact.jeremy.besson@gmail.com

Jean-François Boulicaut
Université de Lyon, CNRS, INRIA
INSA-Lyon, LIRIS Combining, UMR5205, F-69621, France
e-mail: jean-francois.boulicaut@insa-lyon.fr

Tias Guns
Department of Computer Science
Katholieke Universiteit Leuven, Celestijnenlaan 200A, B-3001 Leuven, Belgium
e-mail: tias.guns@cs.kuleuven.be

Siegfried Nijssen
Department of Computer Science
Katholieke Universiteit Leuven, Celestijnenlaan 200A, B-3001 Leuven, Belgium
e-mail: siegfried.nijssen@cs.kuleuven.be

tional data (i.e., collections of transactions that are collections of items). Such data sets are quite common in many different application domains like, e.g., basket data analysis, WWW usage mining, biomedical data analysis. In fact, they correspond to binary relations that encode whether a given set of objects satisfies a given set of Boolean properties or not.

In the last few years, it appears that such 0/1 data mining techniques have reached a kind of ripeness from both an algorithmic and an applicative perspective. It is now possible to process large amounts of data to reveal, for instance, unexpected associations between subsets of objects and subsets of properties which they tend to satisfy. An important breakthrough for the frequent set mining technology and its multiple uses has been the understanding of efficient mechanisms for computing the so-called condensed representations on the one hand, and the huge research effort on safe pruning strategies when considering user-defined constraints on the other hand.

Inspired by the pioneering contribution [23], frequent closed set mining has been studied extensively by the data mining community (see, e.g., the introduction to the FIMI Workshop [16]). A state-of-the-art algorithm like LCM [27] appears to be extremely efficient and it is now possible, for relevant frequency thresholds, to extract every frequent closed set from either sparse or dense data. The analogy between Formal Concept Analysis (see, e.g., [15]) and frequent closed set mining is well understood and this has motivated the design of new algorithms for computing closed sets and concept lattices. Closed set mining has been also studied as a very nice example of a condensed representation for frequency queries and this topic has motivated quite a large number of contributions the last 5 years (see [10] for a survey). In the same time, the active use of user-defined constraints has been studied a lot (see, e.g., [8, 2, 5]). Most of the recent set pattern mining algorithms can exploit constraints that are not limited to the simple cases of monotonic and/or anti-monotonic ones as described in, for instance, [9]. New concepts have emerged like "flexible constraints" [25], "witnesses" [19] or "soft constraints" [4]. Also, the specific problems of noisy data sets has inspired a constraint-based mining view on fault-tolerance (see, e.g., [24, 28, 3]).

While a few tens of important algorithms have been proposed, we lack a clear abstraction of their principles and implementation mechanisms. We think that a timely challenge is to address this problem. Our objective is twofold. First, we want to elucidate the essence of the already published pattern discovery algorithms which process binary relations. Next, we propose a high-level abstraction of them. To this end, we adopt a constraint programming approach which both suits well with the type of problems we are interested in and can help to identify and to describe all basic steps of the constraint-based mining algorithms. Algorithms are presented without any concern about data structures and optimization issues. We would like to stay along the same lines of [21] which introduced in 1997 the level-wise algorithm. This was an abstraction of severals algorithms already published for frequent itemset mining (typically APRIORI) as well as several works around inclusion dependencies and functional dependencies. The generality of the levelwise algorithm inspired a lot of work concerning the use of the border principle and its relations with classical version spaces, or the identification of fundamental mechanisms of enumeration and pruning strategies. Even though this appears challenging, we consider this paper as a major step towards the definition of a flexible, generic though efficient local pattern mining algorithm.

To illustrate the general approach, we will describe two instances of it: one is a specialized, but sufficiently general pattern mining solver in which pattern mining constraints are the main primitives; the other involves the use of existing constraint programming systems, similar to [13]. Both approaches will be illustrated on a problem of fault tolerant mining to make clear the possible advantages and disadvantages.

The rest of this article is organized as follows. The next section introduces some notations and the needed concepts. Section 3 discusses the principles of several specialized algorithms that have been proposed to support set pattern discovery from 0/1 data. Then, we propose in Section 4 an abstraction of such algorithms. Given such a generalization, both the dedicated solver (Section 5) and an implementation scheme within constraint programming systems (Section 6) are given. Section 7 briefly concludes.

2 General Concepts

Let $\mathcal{T} = \{t_1, \ldots, t_m\}$ and $\mathcal{I} = \{i_1, \ldots, i_n\}$ be two sets of respectively transactions and items. Let $\mathbf{r}$ be a boolean matrix in which $\mathbf{r}_{ti} \in \{0, 1\}$, for $t \in \mathcal{T}$ and $i \in \mathcal{I}$. An illustration is given in Figure 1 for the sets $\mathcal{T} = \{t_1, t_2, t_3, t_4, t_5\}$ and $\mathcal{I} = \{i_1, i_2, i_3, i_4\}$.

	i_1	i_2	i_3	i_4
t_1	1	1	1	1
t_2	0	1	0	1
t_3	0	1	1	0
t_4	0	0	1	0
t_5	0	0	0	0

Fig. 1 Boolean matrix where $\mathcal{T} = \{t_1, t_2, t_3, t_4, t_5\}$ and $\mathcal{I} = \{i_1, i_2, i_3, i_4\}$

In such data sets, we are interested in finding local patterns. A local pattern P is a pair of an itemset and a transaction set, (X, Y), which satisfies user-defined local constraints $\mathcal{C}(P)$. We can consider two types of constraints. First, we have constraints on the *pattern types*, which refers to how the pattern is defined with respect to the input data set. Second, we have constraints on the *pattern form*, which do not take into account the data. Hence the constraint $\mathcal{C}$ can be expressed in the form of a conjunction of two constraints

$\mathcal{C}(P) = \mathcal{C}_{type}(P) \wedge \mathcal{C}_{form}(P)$; $\mathcal{C}_{type}$ and $\mathcal{C}_{form}$ can themselves be conjunctions of constraints, i.e., $\mathcal{C}_{type} \equiv \mathcal{C}_{1-type} \wedge \cdots \wedge \mathcal{C}_{k-type}$ and $\mathcal{C}_{form} \equiv \mathcal{C}_{1-form} \wedge \cdots \wedge \mathcal{C}_{l-form}$.

The typical pattern type constraints are given below.

Definition 1 (Main Pattern Types). Let $P = (X, Y) \in 2^{\mathcal{T}} \times 2^{\mathcal{I}}$ be a pattern.

(a) P is an itemset with its support set, satisfying $\mathcal{C}_{itemset}(P)$, iff the constraint $(X = \{x \in \mathcal{T} \mid \forall y \in Y, \mathbf{r}_{xy} = 1\})$ is satisfied.
(b) P is a maximal itemset, satisfying $\mathcal{C}_{max-itemset}(P)$, iff $\mathcal{C}_{itemset}(P)$ is satisfied and there does not exist any itemset $P' = (X', Y')$ satisfying $\mathcal{C}_{itemset}(P')$ such that $X' \subseteq X$ and $Y \subset Y'$. Note that $\mathcal{C}_{itemset}$ is usually substituted with a conjunction of more complex constraints[1] with respect to additional constraints on the patterns P'.
(c) P is a formal concept, satisfying $\mathcal{C}_{fc}(P)$, iff $\mathcal{C}_{itemset}(P)$ is satisfied and there does not exist any itemset $P' = (X', Y')$ such that $X = X'$ and $Y \subset Y'$.

These constraints are related to each other:

$$\mathcal{C}_{max-itemset}(P) \implies \mathcal{C}_{fc}(P) \implies \mathcal{C}_{itemset}(P)$$

Example 1. Referring to Figure 1, (t_1t_2, i_4) and $(t_1, i_2i_3i_4)$ are examples of itemsets with their support sets in $\mathbf{r}_1$. (t_1t_2, i_2i_4) and (t_1t_3, i_2i_3) are two examples of formal concepts in $\mathbf{r}_1$.

The most well-known form constraints are given in Figure 2. The first one is usually called the minimum frequency constraint on itemsets. The second one imposes that both sets of the pattern have a minimal size. The third one is called the "minimal area constraint" and ensures that extracted patterns cover a minimal number of "1" values of the boolean matrix. The next constraint requires that the mean of positive real values associated to each item of the itemset is greater than a given threshold. Constraint $\mathcal{C}_{membership}$ imposes that patterns contain certain elements, for instance $a \in \mathcal{T}$ and $b \in \mathcal{I}$. Emerging patterns satisfying $\mathcal{C}_{emerging}$ must be frequent with respect to a transaction set and infrequent with respect to another one. $\mathcal{C}_{division}$ is another example of a form constraint.

There is a wide variety of combinations of constraints that one could wish to express. For instance, we may want to find itemsets with their support sets (pattern type), for which the support set is greater than 10% (pattern form); or we may wish to find the formal concepts (pattern type) containing at least 3 items and 4 transactions (pattern form) or a fault-tolerant pattern (pattern type) having an area of size at least 20 (pattern form). Fault-tolerant

[1] Indeed, if we would only consider the $\mathcal{C}_{itemset}$ constraint, the only maximal itemset is the itemset containing all items.

$\mathcal{C}_{form}(X,Y)$		
$\mathcal{C}_{size}$	$\equiv$	$\lvert X\rvert > \alpha$
$\mathcal{C}_{min\_rect}$	$\equiv$	$\lvert X\rvert > \alpha \wedge \lvert Y\rvert > \beta$
$\mathcal{C}_{area}$	$\equiv$	$\lvert X\rvert \times \lvert Y\rvert > \alpha$
$\mathcal{C}_{mean}$	$\equiv$	$\sum_{t\in X} Val^{+}(t)/\lvert X\rvert > \alpha$
$\mathcal{C}_{membership}$	$\equiv$	$a \in X \wedge b \in Y$
$\mathcal{C}_{emerging}$	$\equiv$	$\lvert X \cap E_1\rvert > \alpha \wedge \lvert X \cap E_2\rvert < \beta$
$\mathcal{C}_{division}$	$\equiv$	$\lvert X\rvert/\lvert Y > \alpha$

Fig. 2 Examples of interesting pattern form constraints on patterns.

extensions of formal concepts were previously studied in [30, 12, 20, 3], and will be discussed later in more detail.

As a further example, if the items represent the books of "Antoine De Saint-Exupery" and the transactions people who have read some of these books (a '1' value in the boolean matrix), we may want the groups of at least three people who have read at least three books in common including "The little prince". This extraction task can be declaratively defined by the means of the constraint $\mathcal{C}_{EP} = \mathcal{C}_{type} \wedge \mathcal{C}_{form}$ where $\mathcal{C}_{type} = \mathcal{C}_{fc}$ and $\mathcal{C}_{form} = \mathcal{C}_{min\_rect} \wedge \mathcal{C}_{membership}$ ($|X| > 3$, $|Y| > 3$ and "The little prince" $\in Y$).

For the development of algorithms it is important to study the properties of the constraints. Especially monotonic, anti-monotonic and convertible constraints play a key role in any combinatorial pattern mining algorithm to achieve extraction tractability.

Definition 2 ((Anti)-monotonic Constraints). A constraint $\mathcal{C}(X,Y)$ is said to be anti-monotonic with respect to an argument X iff $\forall X, X', Y$ such that $X \subseteq X'$: $\neg\mathcal{C}(X,Y) \implies \neg\mathcal{C}(X',Y)$. A constraint is monotonic with respect to an argument X iff $\forall X, X', Y$ such that $X \supseteq X'$: $\neg\mathcal{C}(X,Y) \implies \neg\mathcal{C}(X',Y)$ We will use the term "(anti)-monotonic" to refer to a constraint which is either monotonic or anti-monotonic.

Example 2. The constraint $\mathcal{C}_{size}$ is monotonic. Indeed, if a set X does not satisfy $\mathcal{C}_{size}$ then none of its subsets also satisfies it.

Some constraints are neither monotonic nor anti-monotonic, but still have good properties that can be exploited in mining algorithms. One such class is the class of "convertible constraints" that can be used to safely prune a search-space while preserving completeness.

Definition 3 (Convertible Constraints). A constraint is said to be convertible with respect to X iff it is not (anti)-monotonic and if there exists a total order on the domain of X such that if a pattern satisfies the constraint, then every prefix (when sorting the items along the chosen order) also satisfies it. This definition implies that whenever a pattern does not satisfy the constraint, then every other pattern with this pattern as a prefix does not satisfy the constraint either.

Example 3. Constraint $\mathcal{C}_{mean}$ of Figure 2 is a convertible constraint where $Val^+ : \mathcal{T} \rightarrow \mathcal{R}$ associates a positive real value to every transaction. Let the relation order $\leq_{conv}$ such that $\forall t_1, t_2 \in \mathcal{T}$, we have $t_1 \leq_{conv} t_2 \Leftrightarrow Val^+(t_1) \leq Val^+(t_2)$. Thus when the transaction set $\{t_i, t_j, ..., t_k\}$ ordered by $\leq_{conv}$ does not satisfy $\mathcal{C}_{mean}$ then all the ordered transaction sets $\{t_i, t_j, ..., t_l\}$ such that $t_k \leq_{conv} t_l$ do not satisfy $\mathcal{C}_{mean}$ either.

3 Specialized Approaches

Over the years, several algorithms have been proposed to find itemsets under constraints. In this section, we review some important ones. In the next section we will generalize these methods.

In all well-known itemset mining algorithms, it is assumed that the constraint $\mathcal{C}_{itemset}$ must be satisfied. Hence, for any given set of items Y, it is assumed we can unambiguously compute its support set X. Most were developed with unbalanced market-basket data sets in mind, in which transactions contain few items, but items can occur in many transactions. Hence, they search over the space of itemsets Y and propose methods for deriving $support(Y)$ for these itemsets.

The most famous algorithm for mining itemsets with high support is the Apriori algorithm [1]. The Apriori algorithm lists patterns increasing in itemset size. It operates by iteratively generating candidate itemsets and determining their support sets in the data. For an itemset Y candidates of size $|Y| + 1$ are generated by creating sets $Y \cup \{e\}$ with $e \notin Y$. For example, from the itemset $Y = \{i_2, i_3\}$ the itemset candidate $Y' = \{i_2, i_3, i_4\}$ is generated. All candidates of a certain size are collected; the support sets of these candidates are computed by traversing the boolean matrix in one pass. Exploiting the anti-monotonicity of the size constraint, only patterns whose support set exceeds the required minimum size, are extended again, and so on.

While Apriori searches breadth-first, alternative methods, such as Eclat [29] and FPGrowth [17], traverse the search space depth-first, turning the search into an enumeration procedure which has an enumeration search tree. Each node in the enumeration tree of a depth-first algorithm corresponds to a pattern $(support(Y), Y)$. For each node in the enumeration tree we compute a triple $\langle Y, support(Y), CHILD \cup Y\rangle$, where $CHILD$ is the set of items i such that $support(Y \cup \{i\}) \geq size$. Hence, $CHILD$ contains all items i that can be added to Y such that the resulting itemset is still frequent. A child $\langle support(Y'), Y', IN' \cup Y'\rangle$ in the enumeration tree is obtained by (i) adding an item $i \in CHILD$ to Y; (ii) computing the set $X' = support(Y')$; and (iii) computing the set IN' of items i' from the items $i \in IN$ for which $support(Y' \cup \{i\}) \geq size$. The main efficiency of the depth-first algorithms derives from the fact that the sets $support(Y')$ and $CHILD$ can be computed incrementally. In our example, the support set of itemset $Y' = \{i_2, i_3, i_4\}$ can be computed from the support set of $Y = \{i_2, i_3\}$ by $\mathit{support}(Y') = \{t_1, t_3\} \cap \mathit{support}(i_4) = \{t_1, t_3\} \cap \{t_1, t_2\} = \{t_1\}$, hence only scanning $\mathit{support}(i_4)$ (instead of $\mathit{support}(i_2)$, $\mathit{support}(i_3)$ and $\mathit{support}(i_4)$). The CHILD' set is incrementally computed from the CHILD set, as the monotonicity of the size constraint entails that elements in CHILD' must also be in CHILD. Compared to the level-wise Apriori algorithm, depth-first algorithms hence require less memory to store candidates.

The most well-known algorithm for finding formal concepts is Ganter's algorithm [14], which presents the first formal concept mining algorithm based on a depth-first enumeration as well as an efficient way to handle the closure constraint $\mathcal{C}_{fc}$ by enabling "jumps" between formal concepts. Each itemset is represented in the form of a boolean vector. For instance if $|\mathcal{I}| = 4$ then the pattern $(1, 0, 1, 0)$ stands for the pattern $\{i_1, i_3\}$ ("1" for presence and "0" for absence). Formal concepts are enumerated in the lexicographic order of the boolean vectors. For example with three items $\mathcal{I} = \{i_1, i_2, i_3\}$, itemsets are ordered in the following way: $\emptyset$, $\{i_3\}$,$\{i_2\}$, $\{i_2, i_3\}$, $\{i_1\}$, $\{i_1, i_3\}$, $\{i_1, i_2\}$ and $\{i_1, i_2, i_3\}$. Assume that we have given two boolean vectors $A = (a_1, \cdots, a_m)$, $B = (b_1, \cdots, b_m)$, both representing itemsets, then Ganter defines (i) an operator $A_i^+ = (a_1, \cdots, a_{i-1}, 1, 0, \cdots, 0)$; (ii) a relation $A <_i B$ which holds iff $a_i < b_i$ and $\forall j < i : a_j = a_i$; and (iii) an operator $A \oplus i = (A_i^+)''$, where $''$ is the closure operator, which adds items included in all transactions covered by (A_i^+). These operators allow us to enumerate the formal concepts in a given order: the smallest formal concept after A is $A \oplus i$ where i is the largest integer satisfying $A \leq_i A \oplus i$. This principle is called prefix preserving closure (PPC) extension [26].

In the example of Figure 2, if we consider the pattern $\{i_2, i_3\}$ corresponding to the boolean vector $A = (0, 1, 1, 0)$, $i = 1$ is the largest integer satisfying $A \leq_i A \oplus i$. Thus from the itemset $\{i_2, i_3\}$ we can "jump" to the formal concept $\{i_1, i_2, i_3, i_4\} = A \oplus 1$.

A disadvantage with this method is that an itemset A and its successor B in the lexicographic order may be completely different, i.e., $A \cap B = \emptyset$. This means that there is no way to use information of a pattern (A in the example) to compute its successor (B in the example). This may be a problem when we are computing formal concepts in large boolean matrices.

This efficiency problem was addressed in algorithms that combine the frequent itemset mining problem with formal concept analysis (usually called frequent closed itemset miners). Most algorithms for mining closed itemsets borrow ideas both from Ganter's algorithm and depth-first itemset miners like LCM [26]. Similar to depth-first itemset miners, they traverse the search-space depth first, and incrementally compute both support and itemsets; to restrict the search space to formal concepts, they apply Ganter's enumeration order; the main idea is to recursively consider only those candidate itemsets which are PPC extensions.

To address the problem of extracting maximal itemset mining under constraints, the DualMiner algorithm was proposed in [9]. It extends the depth-first frequent itemset mining approach to deal with maximality and monotonic constraints on the itemsets. When we need to satisfy monotonic itemset constraints, no longer every node in the enumeration tree corresponds to a pattern; for instance, if we have the constraint $|Y| \geq 3$, all itemsets of size < 3 are no longer patterns, although we might need to traverse these nodes to reach patterns that do satisfy the constraints. The first modification proposed in DualMiner is to test for each node in the enumeration tree if $Y \cup \mathit{CHILD}$ satisfies the monotonic constraint on the itemset: in our example, assume that $|Y \cup \mathit{CHILD}| < 3$ for a certain node in the enumeration tree, then we no longer need to search further below this node, as none of the itemsets we will be creating can satisfy the monotonic constraint, as they are subsets of $Y \cup \mathit{CHILD}$. To speed-up the search for maximal itemsets a similar observation is used: assume that we find that $Y \cup \mathit{CHILD}$ satisfies the anti-monotonic constraint on the itemsets, then we can skip the enumeration of all itemsets Y' for which $Y \subseteq Y' \subset Y \cup \mathit{CHILD}$, as they cannot be maximal.

One way to think of this algorithm is that every node in its enumeration tree corresponds to a tuple $\langle \mathit{support}(\top_{\mathcal{I}}), \bot_{\mathcal{I}}, \mathit{support}(\bot_{\mathcal{I}}), \top_{\mathcal{I}} \rangle$ such that all patterns (X, Y) found below that node will have $\bot_{\mathcal{I}} \subseteq Y \subseteq \top_{\mathcal{I}}$ and $\mathit{support}(\top_{\mathcal{I}}) \subseteq X \subseteq \mathit{support}(\bot_{\mathcal{I}})$. The pair $\langle \bot_{\mathcal{I}}, \top_{\mathcal{I}} \rangle$ represents a search space (also called a subalgebra, or a sublattice). The set $\top_{\mathcal{I}}$ is defined to be the set $\{i \mid \mathcal{C}(\mathit{support}(\bot_{\mathcal{I}} \cup \{i\}), \bot_{\mathcal{I}} \cup \{i\})\}$, for constraints that are monotonic in X or anti-monotonic in Y. The sets $\top_{\mathcal{I}}$ and $\bot_{\mathcal{I}}$ are used as *witnesses*: we use them to either prune an entire search tree, or to jump to a maximal solution.

This idea was generalized in [19, 25, 6]. In [19] it was shown how to compute witnesses for the more difficult "variance" constraint, a problem that remained opened for several years in the data mining community. Soulet et al. [25] extend the idea to deal with more difficult constraints such as the area constraint, which take into account both support sets and itemsets. For example, if we want to compute all the patterns (X, Y) satisfying $\mathcal{C}_{itemset}$ with an area greater than 3 ($\mathcal{C}_{area}$ where $\alpha = 3$), knowing that $\bot_{\mathcal{I}} \subseteq Y \subseteq \top_{\mathcal{I}}$ and hence $\mathit{support}(\top_{\mathcal{I}}) \subseteq X \subseteq \mathit{support}(\bot_{\mathcal{I}})$, then we can bound the area of (X, Y) by $|\mathit{support}(\top_{\mathcal{I}})| \times |\bot_{\mathcal{I}}| \leq |X| \times |Y| \leq |\mathit{support}(\bot_{\mathcal{I}})| \times |\top_{\mathcal{I}}|$. If $|\mathit{support}(\bot_{\mathcal{I}})| \times |\top_{\mathcal{I}}| < 4$, any pattern of the current search space has an area lower than 4, and one can safely stop considering itemsets below (X, Y). A more sophisticated extension was proposed by Bonchi et al. based on the ExAnte property [6, 7]. The ExAnte property states that if a transaction does not satisfy an easily computable monotonic itemset constraint (such as that the itemset has at least size 3), then this transaction can never be in $\mathit{support}(X)$ (as the transaction can only be in the support set of itemsets with less than 3 items). Hence, we can remove this transaction from consideration. The consequence of this removal is that we are reducing the support counts of all items included in the transaction, which may turn some of them infrequent. Removing infrequent items from consideration, some transactions may no longer satisfy the constraint on the itemsets, and can be removed again; this procedure can be repeated till we reach a fixed point in which both support sets and itemsets do not change any more.

Originally, this was proposed as a pre-processing procedure that can be applied on any dataset to obtain a smaller dataset [6]. However, later it was observed that we can perform such pruning in every node of the enumeration tree of a depth-first itemset miner [7]. Essentially, in every node of the enumeration tree, we have a tuple

$$\langle \mathit{support}(\top_{\mathcal{I}}), \bot_{\mathcal{I}}, \top_{\mathcal{T}}, \top_{\mathcal{I}}, \rangle$$

where $\top_{\mathcal{T}}$ denotes the set of transactions that can still be covered by a pattern. Compared to the DualMiner and other itemset miners, the key observation is that we also evaluate constraints on the transactions, and allow these evaluations to change the set $\top_{\mathcal{I}}$; no longer do we implicitly assume the set $\top_{\mathcal{T}}$ to equal $\mathit{support}(\bot_{\mathcal{I}})$.

4 A Generalized Algorithm

We propose to generalize the various approaches that have been sketched in the previous section. It is based on a depth-first search in which every node of the enumeration tree has a tuple

$$SP = \langle \bot, \top \rangle = \langle \bot_{\mathcal{T}} \cup \bot_{\mathcal{I}}, \top_{\mathcal{T}} \cup \top_{\mathcal{I}} \rangle,$$

such that below this node we will only find patterns $P = (X, Y)$ in which $\bot_{\mathcal{T}} \subseteq X \subseteq \top_{\mathcal{T}}$ and $\bot_{\mathcal{I}} \subseteq Y \subseteq \top_{\mathcal{I}}$. We will abbreviate this to $P \in SP$; one can say that sets $\bot_{\mathcal{T}}$ and $\top_{\mathcal{T}}$ define the domain of X, and sets $\bot_{\mathcal{I}}$ and $\top_{\mathcal{I}}$ the domain of Y.

Constraints express properties that should hold between X and Y. During the search, this means that if we change the domain for X, this can have an effect on the possible domain for Y, and the other way around. This idea of *propagating* changes in the domain of one set to the domain of another set, is essential in *constraint programming.* The general outline of the search that we wish to perform for itemset mining is given in Figure 3.

Figure 3 presents a skeleton of a binary depth-first search algorithm. According to the data set and the constraint $\mathcal{C}$, the domains are first of all reduced through propagation. If the domain is still consistent then the enumeration process keeps going. If a solution is found, it is printed. Otherwise the algorithm selects an element to be enumerated and generates two new nodes (with the function **Search**). Finally, the algorithm is recursively called on the two newly generated nodes. Let us now study in more details what we mean by propagation and consistency.

```
Search(A search-space $\langle \bot, \top \rangle$, a data set $\mathbf{r}$ and a constraint $\mathcal{C}$ on $2^{\mathcal{T}} \times 2^{\mathcal{I}}$)
repeat for variables $e \in \mathcal{I} \cup \mathcal{T}$ with $e \notin \bot$ and $e \in \top$ till fixpoint:      (Propagation)
    if $\neg Upper_{\mathcal{C}}(\langle \bot \cup \{e\}, \top \rangle)$ then $\langle \bot, \top \rangle \leftarrow \langle \bot, \top \setminus \{e\} \rangle$
    if $\neg Upper_{\mathcal{C}}(\langle \bot, \top \setminus \{e\} \rangle)$ then $\langle \bot, \top \rangle \leftarrow \langle \bot \cup \{e\}, \top \rangle$
if $Upper_{\mathcal{C}}(\langle \bot, \top \rangle)$ then      (Consistency check)
  if $\bot = \top$ then
    Print$\langle \bot, \top \rangle$      (Solution found)
  else
    Let $e \in \mathcal{I} \cup \mathcal{T}$ with $e \notin \bot$ and $e \in \top$
    Search($\langle \bot, \top \setminus \{e\} \rangle$,$\mathbf{r}$,$\mathcal{C}$)
    Search($\langle \bot \cup \{e\}, \top \rangle$,$\mathbf{r}$,$\mathcal{C}$)
```

Fig. 3 Skeleton of a binary backtracking algorithm

Definition 4 (Constraint Upper bounds). Let SP be the search-space and $\mathcal{C}$ a constraint over $2^{\mathcal{T}} \times 2^{\mathcal{I}}$. Then an upper-bound of $\mathcal{C}$ on SP is a predicate $Upper_{\mathcal{C}}(SP)$ such that if there exists $P \in SP$ for which $\mathcal{C}(P)$ is true, then predicate $Upper_{\mathcal{C}}(SP)$ is true; furthermore, if $\bot = \top$, it should hold that $Upper_{\mathcal{C}}(SP) = \mathcal{C}(\bot)$. Informally, if SP violates an upper-bound of $\mathcal{C}$ then SP is not consistent, i.e., no valid pattern can be generated from this search-space.

From constraint upper-bounds $Upper_{\mathcal{C}}$ and search-space lower- and upper-bounds $SP = \langle \bot, \top \rangle$, propagation can be applied thanks to the following observation:

- if an element $e \in (\top \setminus \bot)$, once added to the lower-bound of its set, violates the predicate $Upper_{\mathcal{C}}(\langle \bot \cup \{e\}, \top \rangle)$, then $\mathcal{C}$ can never be satisfied if e is included in the solution and e should be remove from the upper-bound of the set.
- if an element $e \in (\top \setminus \bot)$, once removed from the upper-bound of its set, violates the predicate $Upper_{\mathcal{C}}(\langle \bot, \top \setminus \{e\} \rangle)$, then $\mathcal{C}$ can never be satisfied if e is not included in the solution, and e should be added to the lower-bound of the set.

Our algorithm generalizes methods such as Eclat, DualMiner and ExAnte. It maintains all bounds that are also maintained in such algorithms; if a constraint $\mathcal{C}_{itemset}$ is enforced, its propagation ensures that the bounds for the transaction set correspond to the appropriate support set, as in these algorithms. The iterative application of propagation is borrowed from the ExAnte algorithm if monotonic constraints on the itemsets are used.

Even though the algorithm in Figure 3 shows how we would like to perform the search, there are multiple ways of formalizing itemset mining problems and implementing propagation, pruning and search. We present two ways to deal with such issues in the next two sections.

5 A Dedicated Solver

5.1 Principles

Our first option is to build a dedicated, but still generic enough algorithm for itemset mining. In such a system, the key idea is that the system provides for the search, as indicated before, but the user has the ability to plug in algorithms for evaluating the constraints. These algorithms can be implemented in arbitrary programming languages, and our main problem here is to decide how the search procedure may exploit such plug in algorithms.

Let us first consider the simple case, in which we assume that we have an algorithm for evaluating a constraint $\mathcal{C}(X, Y)$ which is (anti-)monotonic in each of its parameters.

Definition 5 (Upperbound for (Anti-)Monotonic Constraints). Let $\mathcal{C}$ be an (anti)-monotonic constraint both on itemsets and transaction sets, i.e., the so-called bisets. The following is a valid upper bound:

$$\mathit{Upper}_{\mathcal{C}}(SP) = \mathcal{C}(M_1(SP), M_2(SP)),$$

where $M_1(SP)$ equals $\top_{\mathcal{T}}$ (resp. $\bot_{\mathcal{T}}$) if $\mathcal{C}$ is monotonic (resp. anti-monotonic) on the transaction set. $M_2(SP)$ is defined similarly for itemsets.

Example 4. The constraint $\mathcal{C}_{division}(X, Y) \equiv |X|/|Y| > \alpha$ is monotonic on X and anti-monotonic on Y with respect to the inclusion order. Indeed let $X_1 \subseteq X_2 \in \mathcal{T}$ and $Y_1 \subseteq Y_2 \in \mathcal{I}$, we have $\mathcal{C}_{division}(X_1, Y) \Rightarrow \mathcal{C}_{division}(X_2, Y)$ and $\mathcal{C}_{division}(X, Y_2) \Rightarrow \mathcal{C}_{division}(X, Y_1)$. Finally, the upperbound of $\mathcal{C}_{division}$ is $\mathit{Upper}_{\mathcal{C}_{division}}(SP) = |\top_{\mathcal{T}}|/|\bot_{\mathcal{I}}| > \alpha$.

It is possible to exploit such an observation to call the constraint evaluation algorithm when needed. We can generalize this to constraints which are not monotonic or anti-monotonic in each of its parameters. Let us start with the definition of a function $\mathcal{P}_{\mathcal{C}}$ which is a simple rewriting of $\mathcal{C}$.

Definition 6 ($\mathcal{P}_{\mathcal{C}}$). Let $\mathcal{C}(X, Y)$ be a constraint on $2^{\mathcal{T}} \times 2^{\mathcal{I}}$. We denote by $\mathcal{P}_{\mathcal{C}}$ the constraint obtained from $\mathcal{C}$ by substituting each instance of X (resp. Y) in $\mathcal{C}$ with an other parameter X_i (resp. Y_i) in the constraint $\mathcal{P}_{\mathcal{C}}$.

For example, if we want to compute bisets satisfying $\mathcal{C}_{mean}$, i.e., a mean above a threshold α on a criterion $Val^+ : \mathcal{T} \to \mathbb{R}^+$:

$$\mathcal{C}_{mean}(X) \equiv \frac{\sum_{x \in X} Val^+(x)}{|X|} > \alpha.$$

The argument X appears twice in the expression of $\mathcal{C}_{mean}$. To introduce the notion of piecewise monotonic constraint, we have to rewrite such constraints using a different argument for each occurrence of the same argument. For example, the previous constraint is rewritten as:

$$\mathcal{P}_{\mathcal{C}_{mean}}(X_1, X_2) \equiv \frac{\sum_{x \in X_1} Val^+(x)}{|X_2|} > \alpha.$$

Another example is the constraint specifying that bisets must contain a proportion of a given biset (E, F) larger than a threshold α:

$$\mathcal{C}_{intersection}(X, Y) \equiv \frac{|X \cap E| \times |Y \cap F|}{|X| \times |Y|} > \alpha.$$

This constraint is rewritten as

$$\mathcal{P}_{\mathcal{C}_{intersection}}(X_1, X_2, Y_1, Y_2) \equiv \frac{|X_1 \cap E| \times |Y_1 \cap F|}{|X_2| \times |Y_2|} > \alpha$$

We can now define the class of piecewise (anti)-monotonic constraints for which we can define an $Upper_{\mathcal{C}}$ predicate, which allows us to push the constraint in the generic algorithm:

Definition 7 (Piecewise (Anti)-Monotonic Constraint). A constraint $\mathcal{C}$ is piecewise (anti)-monotonic if its associated constraint $\mathcal{P}_{\mathcal{C}}$ is either monotonic or anti-monotonic on each of its arguments. We denote by $\mathcal{X}_m$ (respectively $\mathcal{Y}_m$) the set of arguments X_i (resp. Y_i) of $\mathcal{P}_{\mathcal{C}}$ for which $\mathcal{P}_{\mathcal{C}}$ is monotonic. In the same way $\mathcal{X}_{am}$ (respectively $\mathcal{Y}_{am}$) denotes the set of arguments X_i (resp. Y_i) of $\mathcal{P}_{\mathcal{C}}$ for which $\mathcal{P}_{\mathcal{C}}$ is anti-monotonic.

Example 5. The constraint $\mathcal{C}_{mean}(X) \equiv \sum_{i \in X} Val^+(i)/|X| > \alpha$, which is not (anti)-monotonic, is piecewise (anti)-monotonic. We can check that $\mathcal{P}_{\mathcal{C}_{mean}} \equiv \sum_{i \in X_1} Val^+(i)/|X_2| > \alpha$ is (anti)-monotonic for each of its arguments, i.e., X_1 and X_2.

We can now define upper-bounds of piece-wise (anti)-monotonic constraints. An upper-bound of a piecewise (anti)-monotonic constraint $\mathcal{C}$ is:

$$Upper_{\mathcal{C}}(SP) = \mathcal{P}_{\mathcal{C}}(P_1, \cdots, P_m),$$

where

$$\begin{aligned} P_i &= \top_{\mathcal{T}} \;\text{ if } P_i \in \mathcal{X}_m \\ P_i &= \bot_{\mathcal{T}} \;\text{ if } P_i \in \mathcal{X}_{am} \\ P_i &= \top_{\mathcal{I}} \;\text{ if } P_i \in \mathcal{Y}_m \\ P_i &= \bot_{\mathcal{I}} \;\text{ if } P_i \in \mathcal{Y}_{am} \end{aligned}$$

Example 6. We have $\mathcal{P}_{\mathcal{C}_1}(X_1, X_2) \equiv \sum_{i \in X_1} Val^+(i)/|X_2| > \alpha$ where $\mathcal{X}_m = \{X_1\}$ and $\mathcal{X}_{am} = \{X_2\}$. Thus we obtain $Upper(\mathcal{C}_1) \equiv$

$$\sum_{i \in \top_{\mathcal{T}}} \frac{Val^{+}(i)}{|\bot_{\mathcal{T}}|} > \alpha.$$

For $\mathcal{P}_{\mathcal{C}_2}(X_1, X_2, Y_1) \equiv |X_1 \cup E| * |Y_1|/|X_2| > \alpha$, we have $\mathcal{X}_m = \{X_1\}$, $\mathcal{Y}_m = \{Y_1\}$ and $\mathcal{X}_{am} = \{X_2\}$. Thus an upper-bound of $\mathcal{C}_2$ is $Upper(\mathcal{C}_2) \equiv$

$$\frac{|\top_{\mathcal{T}} \cup E| * |\top_{\mathcal{I}}|}{|\bot_{\mathcal{T}}|} > \alpha.$$

In [25], the authors present a method to compute the same upper-bounds of constraints, but built from a fixed set of primitives. Notice also that [11] provides an in-depth study of piecewise (anti-)monotonicity impact when considering the more general setting of arbitrary $n-$ary relation mining.

Overall, in this system, a user would implement a predicate $\mathcal{P}_{\mathcal{C}}$, and specify for each parameter of this predicate if it is monotone or anti-monotone.

5.2 Case study on formal concepts and fault-tolerant patterns

Let us now illustrate the specialized approach on concrete tasks like formal concept analysis and fault-tolerant pattern mining. In the next section, the same problems will be addressed using an alternative approach.

We first show that $\mathcal{C}_{fc}$, the constraint which defines formal concepts, is a piece-wise (anti)-monotonic constraint. Then, we introduce a new fault-tolerant pattern type that can be efficiently mined thanks to the proposed framework.

We can rewrite $\mathcal{C}_{fc}$ (see Section 2) to get $\mathcal{P}_{\mathcal{C}_{fc}}$:

$\mathcal{P}_{\mathcal{C}_{fc}}(X_1, X_2, X_3, Y_1, Y_2, Y_3) =$

$$\begin{array}{l} \bigwedge_{t \in X_1} \bigwedge_{i \in Y_1} \mathbf{r}_{ti} \\ \bigwedge_{t \in \mathcal{T} \setminus X_2} \bigvee_{i \in Y_2} \neg \mathbf{r}_{ti} \\ \bigwedge_{i \in \mathcal{I} \setminus Y_3} \bigvee_{t \in X_3} \neg \mathbf{r}_{ti} \end{array}$$

Analysing the monotonicity of $\mathcal{P}_{\mathcal{C}_{CF}}$ we can check that $\mathcal{C}_{fc}$ is a piecewise (anti)-monotonic constraint where $\mathcal{X}_m = \{X_2\}$, $\mathcal{X}_{am} = \{X_1, X_3\}$, $\mathcal{Y}_m = \{Y_3\}$ and $\mathcal{Y}_{am} = \{Y_1, Y_2\}$. Finally we can compute an upper-bound of $\mathcal{C}_{fc}$:

$Upper_{\mathcal{C}_{fc}}(SP) = \mathcal{P}_{\mathcal{C}_{CF}}(\bot_{\mathcal{T}}, \top_{\mathcal{T}}, \bot_{\mathcal{T}}, \bot_{\mathcal{I}}, \bot_{\mathcal{I}}, \top_{\mathcal{I}}) =$

$$\begin{array}{l} \bigwedge_{t \in \bot_{\mathcal{T}}} \bigwedge_{i \in \bot_{\mathcal{I}}} \mathbf{r}_{ti} \\ \bigwedge_{t \in \mathcal{T} \setminus \top_{\mathcal{T}}} \bigvee_{i \in \bot_{\mathcal{I}}} \neg \mathbf{r}_{ti} \\ \bigwedge_{i \in \mathcal{I} \setminus \top_{\mathcal{I}}} \bigvee_{t \in \bot_{\mathcal{T}}} \neg \mathbf{r}_{ti} \end{array}$$

Besides the well-known and well-studied formal concepts, there is an important challenge which concerns the extraction of combinatorial fault-tolerant patterns (see, e.g., [24, 3]). The idea is to extend previous patterns to enable some false values (seen as exceptions) in patterns. We present here the pattern type introduced in [3].

Definition 8 (Fault-tolerant pattern). A biset (X, Y) is a fault-tolerant pattern iff it satisfies the following constraint $\mathcal{C}_{DRBS}$:
$\mathcal{Z}_\mathcal{T}(t, Y) = |\{i \in Y \mid \neg \mathbf{r}_{ti}\}|$
$\mathcal{Z}_\mathcal{I}(i, X) = |\{t \in X \mid \neg \mathbf{r}_{ti}\}|$

$$\mathcal{C}_{DRBS}(X, Y) \equiv \begin{cases} \bigwedge_{t \in X} \mathcal{Z}_\mathcal{T}(t, Y) \leq \alpha \\ \bigwedge_{i \in Y} \mathcal{Z}_\mathcal{I}(i, X) \leq \alpha \\ \bigwedge_{t \in \mathcal{T} \setminus X} \bigwedge_{t' \in X} \mathcal{Z}_\mathcal{T}(t', Y) \leq \mathcal{Z}_\mathcal{T}(t, Y) \\ \bigwedge_{i \in \mathcal{I} \setminus Y} \bigwedge_{i' \in Y} \mathcal{Z}_\mathcal{I}(i', X) \leq \mathcal{Z}_\mathcal{I}(i, X) \end{cases}$$

α stands for the maximal number of tolerated false values per row and per column in the pattern. The two last constraints ensure that elements not included in the patterns contain more false values than those included.

Example 7. $(t_1t_2t_3, i_2i_3i_4)$ and $(t_1t_2t_3t_4, i_2i_3)$ are examples of fault-tolerant patterns in $\mathbf{r}_1$ with $\alpha = 1$.

We now need to check whether $\mathcal{C}_{DRBS}$ can be exploited within our generic algorithm, i.e., whether it is a piece-wise anti-monotonic constraint. Applying the same principles as described before, we can compute the predicate $\mathcal{P}_{\mathcal{C}_{DRBS}}$ as following:
$\mathcal{P}_{\mathcal{C}_{DRBS}}(X_1, \cdots, X_6, Y_1, \cdots, Y_6) =$

$$\begin{array}{l} \bigwedge_{t \in X_1} \mathcal{Z}_\mathcal{T}(t, Y_1) \leq \alpha \\ \bigwedge_{i \in Y_2} \mathcal{Z}_\mathcal{I}(i, X_2) \leq \alpha \\ \bigwedge_{t \in \mathcal{T} \setminus X_3} \bigwedge_{t' \in X_4} \mathcal{Z}_\mathcal{T}(t', Y_3) \leq \mathcal{Z}_\mathcal{T}(t, Y_4) \\ \bigwedge_{i \in \mathcal{I} \setminus Y_5} \bigwedge_{i' \in Y_6} \mathcal{Z}_\mathcal{I}(i', X_5) \leq \mathcal{Z}_\mathcal{I}(i, X_6) \end{array}$$

According to Definition 7, $\mathcal{C}_{DRBS}$ is a piecewise (anti)-monotonic constraint where $\mathcal{X}_m = \{X_3, X_6\}$, $\mathcal{X}_{am} = \{X_1, X_2, X_4, X_5\}$, $\mathcal{Y}_m = \{Y_4, Y_5\}$ and $\mathcal{Y}_{am} = \{Y_1, Y_2, Y_3, Y_6\}$.

Finally we can compute an upper-bound of $\mathcal{C}_{DRBS}$:
$Upper_{\mathcal{C}_{fc}}(SP) =$

$$\begin{array}{l} \bigwedge_{i \in \bot_\mathcal{T}} \mathcal{Z}_\mathcal{T}(i, \bot_\mathcal{I}) \leq \alpha \\ \bigwedge_{i \in \bot_\mathcal{I}} \mathcal{Z}_\mathcal{I}(i, \bot_\mathcal{T}) \leq \alpha \\ \bigwedge_{t \in \mathcal{T} \setminus \top_\mathcal{T}} \bigwedge_{t' \in \bot_\mathcal{T}} \mathcal{Z}_\mathcal{T}(t', \bot_\mathcal{I}) \leq \mathcal{Z}_\mathcal{T}(t, \top_\mathcal{I}) \\ \bigwedge_{i \in \mathcal{I} \setminus \top_\mathcal{I}} \bigwedge_{i' \in \bot_\mathcal{I}} \mathcal{Z}_\mathcal{T}(i', \bot_\mathcal{T}) \leq \mathcal{Z}_\mathcal{I}(i, \top_\mathcal{T}) \end{array}$$

6 Using Constraint Programming Systems

6.1 Principles

An alternative approach is to require that the user specifies constraints in a *constraint programming* language. The constraint programming system is responsible for deriving bounds for the specified constraints. This approach was taken in [13], where it was shown that many itemset mining tasks can be specified using primitives that are available in off-the-shelf constraint programming systems. The essential constraints that were used are the so-called reified summation constraints:

$$(V' = 1) \Leftarrow \sum_k \alpha_k V_k \geq \theta \tag{1}$$

and

$$(V' = 1) \Rightarrow \sum_k \alpha_k V_k \geq \theta, \tag{2}$$

where V', $V_1 \ldots V_n$ are variables with domains $\{0,1\}$ and α_k is a constant for each variable V_k within this constraint.

The essential observation in this approach is that an itemset Y can be represented by a set of boolean variables I_i where $I_i = 1$ iff item $i \in Y$. Similarly, we can represent a transaction set X using a set of boolean variables T_t where $T_t = 1$ iff $t \in X$.

For instance, the $\mathcal{C}_{itemset}$ constraint can be specified using the conjunction of the following constraints:

$$T_t = 1 \Leftrightarrow \sum_{i \in \mathcal{I}} I_i(1 - \mathbf{r}_{ti}) = 0, \qquad \text{for all} \qquad t \in \mathcal{T}$$

This constraint states that a transaction t is in the support set of an itemset if and only if all items in the itemset ($I_i = 1$) are not missing in the transaction ($(1 - \mathbf{r}_{ti}) = 0$).

As it turns out, many constraint programming systems by default provide propagators for these reified summation constraints, by maintaining domains for boolean variables instead of domains for itemsets and transactions. Let $\perp_V$ denote the lowest element in the domain of a variable, and $\top_V$ denote the highest element. Then for the reified summation constraint of equation (2) a propagor computes whether the following condition is true:

$$\sum_{\alpha_k < 0} \alpha_k \perp_{V_k} + \sum_{\alpha_k > 0} \alpha_k \top_{V_k} < \theta;$$

if this condition holds, the sum cannot reach the desired value even in the most optimistic case, and hence the precondition $V' = 1$ cannot be true. Consequently value 1 is removed from the domain of variable V'.

In our running example, if we have transaction 3 with items $\{i_2, i_3\}$, this transaction is represented by the constraint

$$T_3 = 1 \Leftrightarrow I_1 + I_4 = 0.$$

If we set the domain of item I_1 to 1 (or, equivalently, include item i_1 in $\bot_{\mathcal{I}}$), this constraint will be false for $T_3 = 1$. Hence, the evaluation of $Upper_{\mathcal{C}_{itemset}}$ when $t_3 \in \bot_{\mathcal{T}}$ is false, and transaction t_3 will be removed from the domain of $\top_{\mathcal{T}}$.

Consequently, by formalizing the $\mathcal{C}_{itemset}$ constraint using reified implications, we achieve the propagation that we desired in our generalized approach. The search, the propagators and the evaluation of constraints are provided by the constraint programming system; however, the constraints should be specified in the constraint programming language of the system, such as the reified summation constraint.

6.2 Case study on formal concepts and fault-tolerant patterns

Let us reconsider the constraints proposed in Section 5.2, starting with the constraints that define the formal concepts. Below we show that each of these constraints can be rewritten to an equivalent reified summation constraint:

$$\begin{aligned}
\bigwedge_{i \in \mathcal{I} \setminus I} \bigvee_{t \in \mathcal{T}} \neg \mathbf{r}_{ti} &\Leftrightarrow (\forall i \in \mathcal{I} : (\neg I_i) \Rightarrow (\exists t : T_t \wedge \neg \mathbf{r}_{ti})) \\
&\Leftrightarrow \left(\forall i \in \mathcal{I} : (I_i = 0) \Rightarrow \left(\sum_t T_t (1 - \mathbf{r}_{ti}) > 0 \right) \right) \\
&\Leftrightarrow \left(\forall i \in \mathcal{I} : (I_i = 1) \Leftarrow \left(\sum_t T_t (1 - \mathbf{r}_{ti}) = 0 \right) \right)
\end{aligned}$$

$$\bigwedge_{t \in \mathcal{T} \setminus T} \bigvee_{i \in \mathcal{I}} \neg \mathbf{r}_{ti} \Leftrightarrow \left(\forall t \in \mathcal{T} : (T_t = 1) \Leftarrow \left(\sum_i I_i (1 - \mathbf{r}_{ti}) = 0 \right) \right)$$

$$\begin{aligned}\bigwedge_{t\in\mathcal{T}\setminus T}\bigwedge_{i\in\mathcal{I}}\mathbf{r}_{ti} &\Leftrightarrow (\forall t\in\mathcal{T}, i\in\mathcal{I}: \neg T_t \vee \neg I_i \vee \mathbf{r}_{ti})\\ &\Leftrightarrow (\forall t\in\mathcal{T}: \neg T_t \vee (\forall i\in\mathcal{I}: \neg I_i \vee \mathbf{r}_{ti}))\\ &\Leftrightarrow (\forall t\in\mathcal{T}: T_t \Rightarrow (\forall i\in\mathcal{I}: \neg(I_i \wedge \neg\mathbf{r}_{ti})))\\ &\Leftrightarrow \left(\forall t\in\mathcal{T}: (T_t=1) \Rightarrow \left(\sum_i I_i(1-\mathbf{r}_{ti})=0\right)\right)\end{aligned}$$

This rewrite makes clear that we can also formulate the formal concept analysis problem in constraint programming systems. The bounds computed by the CP system correspond to those computed by the specialized approach, and the propagation is hence equivalent.

The second problem that we consider is that of mining fault-tolerant formal concepts. We can observe that

$$\mathcal{Z}_{\mathcal{T}}(t,Y)=\sum_i I_i(1-\mathbf{r}_{ti})$$

and

$$\mathcal{Z}_{\mathcal{I}}(i,X)=\sum_t T_t(1-\mathbf{r}_{ti}).$$

Hence,

$$\bigwedge_{t\in X}\mathcal{Z}_{\mathcal{T}}(t,Y)\le\alpha \Leftrightarrow \left(\forall t\in\mathcal{T}: T_t=1 \Rightarrow \sum_i I_i(1-\mathbf{r}_{ti})\le\alpha\right)$$

and

$$\bigwedge_{i\in Y}\mathcal{Z}_{\mathcal{I}}(i,X)\le\alpha \Leftrightarrow \left(\forall i\in\mathcal{I}: I_i=1 \Rightarrow \sum_t T_t(1-\mathbf{r}_{ti})\le\alpha\right).$$

Note that these formulas are generalizations of the formulas that we developed for the traditional formal concept analysis, the traditional case being $\alpha=0$.

We can also reformulate the other formulas of fault-tolerant itemset mining.

$$\bigwedge_{t\in\mathcal{T}\setminus X}\bigwedge_{t'\in X}\mathcal{Z}_{\mathcal{T}}(t',Y)\le\mathcal{Z}_{\mathcal{T}}(t,Y)\Leftrightarrow$$
$$\left(\forall t,t'\in\mathcal{T}: (T_t=0\wedge T_{t'}=1)\Leftrightarrow \sum_i I_i(1-\mathbf{r}_{t'i})\le\sum_i I_i(1-\mathbf{r}_{ti})\right)$$

However, this formulation yields a number of constraints that is quadratic in the number of transactions. Additionally, it is not in the desired form with one variable on the left-hand side.

A formulation with a linear number of constraints can be obtained by rewriting the problem further.

Let us define an additional constraint over an additional variable $\beta_{\mathcal{T}}$, as follows:

$$\beta_{\mathcal{T}} = \max_t \sum_i I_i(1 - \mathbf{r}_{ti})T_t,$$

which corresponds to the maximum number of 1s missing within one row of the formal concept. Then the following linear set of constraints is equivalent to the previous quadratic set:

$$\forall t \in \mathcal{T} : T_t = 1 \Leftarrow \sum_i I_i(1 - \mathbf{r}_{ti}) \leq \beta_{\mathcal{T}};$$

as we can see, this constraint is very similar to the constraint for usual formal concepts, the main difference being that $\beta_{\mathcal{T}}$ is not a constant, but a variable whose domain needs to be computed. Most constraint programming systems provide the primitives that are required to compute the domain of $\beta_{\mathcal{T}}$.

Observe that adding the reverse implication would be redundant, given how $\beta_{\mathcal{T}}$ is defined.

To enforce sufficient propagation, it may be useful to pose additional, redundant constraints:

$$\beta'_{\mathcal{T}} = \min_t \sum_i I_i(1 - \mathbf{r}_{ti}(1 - T_t))$$

in conjunction with

$$\forall t \in \mathcal{T} : T_t = 1 \Rightarrow \sum_i I_i(1 - \mathbf{r}_{ti}) \leq \beta'_{\mathcal{T}}.$$

This constraint considers the number of 1s missing in rows which are (certainly) not part of the formal concept. Its propagators ensure that we can also determine that certain transactions should not be covered in order to satisfy the constraints.

Similarly, we can express the constraints over items. Our overall set of constraints becomes:

$$\begin{aligned}
\forall t \in \mathcal{T} : T_t = 1 &\Leftarrow \sum_i I_I(1 - \mathbf{r}_{ti}) \leq \beta_{\mathcal{T}} \\
\forall t \in \mathcal{T} : T_t = 1 &\Rightarrow \sum_i I_i(1 - \mathbf{r}_{ti}) \leq \min(\alpha, \beta'_{\mathcal{T}}) \\
\forall i \in \mathcal{I} : I_i = 1 &\Leftarrow \sum_t T_t(1 - \mathbf{r}_{ti}) \leq \beta_{\mathcal{I}} \\
\forall i \in \mathcal{I} : I_i = 1 &\Rightarrow \sum_t T_t(1 - \mathbf{r}_{ti}) \leq \min(\alpha, \beta'_{\mathcal{I}}) \\
\beta_{\mathcal{T}} &= \max_t \sum_i I_i(1 - \mathbf{r}_{ti})T_t \\
\beta'_{\mathcal{T}} &= \min_t \sum_i I_i(1 - \mathbf{r}_{ti}(1 - T_t)) \\
\beta_{\mathcal{I}} &= \max_i \sum_t T_t(1 - \mathbf{r}_{ti})I_i \\
\beta'_{\mathcal{I}} &= \min_i \sum_t T_t(1 - \mathbf{r}_{ti}(1 - I_i))
\end{aligned}$$

Clearly, implementing these fault tolerance constraints using CP systems requires the use of a large number of lower-level constraints, such as reified sum constraints, summation constraints, minimization constraints and maximization constraints, which need to be provided by the CP system. To add these constraints in a specialized system, they need to be implemented in the lower-level language in which the system is implemented itself.

7 Conclusions

Over the years many specialized constraint-based mining algorithms have been proposed, but a more general perspective is overall missing. In this work we studied the formalization of fundamental mechanisms that have been used in various itemset mining algorithms and aimed to describe high-level algorithms without any details about data structures and optimization issues.

Our guiding principles in this study were derived from the area constraint programming. Key ideas in constraint programming are declarative problem specification and constraint propagation. To allow for declarative problem specification, constraint programming systems provide users a modeling language with basic primitives such as inequalities, sums, and logic operators. For each of these primitives, the system implements propagators; propagators can be thought of as algorithms for computing how the variables in a constraint interact with each other.

Within this general framework, there are still many choices that can be made. We investigated two options. In the first option, we developed a methodology in which data mining constraints are added as basic primitives to a specialized CP system. Advantages of this approach are that users do not need to study lower level modeling primitives (such as summation constraints), that users are provided with a clear path for adding primitives to the system, and that it is possible to optimize the propagation better. To simplify the propagation for new constraints, we introduced the class of piecewise (anti-)monotonic constraints. For constraints within this class it is not needed that a new propagator is introduced in the system; it is sufficient to implement an algorithm for evaluating the constraint. This simplifies the extension of the specialized system with new constraints significantly and makes it possible to add constraints within this class in an almost declarative fashion. However, extending the system with other types of constraints is a harder task that requires further study.

The second option is to implement data mining constraints using lower level modeling primitives provided by existing CP systems. The advantage of this approach is that it is often not necessary to add new primitives to the CP system itself; it is sufficient to formalize a problem using a set of lower level modeling primitives already present in the system. It is also clear how different types of constraints can be combined and how certain non piecewise monotonic constraints can be added. The disadvantage is that it is less clear how to optimize the constraint evaluation, if needed, or how to add constraints that cannot be modeled using the existing primitives in the CP system. This means that the user may still have to implement certain constraints in a lower level programming language. Furthermore, the user needs to have a good understanding of lower level primitives available in CP systems and needs to have a good understanding of the principles of propagation, as principles such as piecewise monotonicity are not used.

Comparing these two approaches, we can conclude that both have advantages and disadvantages; it is likely to depend on the requirements of the user which one is to be preferred.

As indicated, there are several further possibilities for extending this work. On the systems side, one could be interested in bridging the gap between these two approaches and build a hybrid approach that incorporates the specialized approach in a general system. This may allow to optimize the search procedure better, where needed.

On the problem specification side, we discussed here only how to apply both approaches to pattern mining. It may be of future interest to study alternative problems, ranging from pattern mining problems such as graph mining, to more general problems such as clustering. Also in these problem settings, it is likely that a hybrid approach will be needed that combines the general approach of constraint programming with more efficient algorithms developed in recent years for specialized tasks. We hope that this work provides inspiration for the development of these future approaches.

Acknowledgments. This work has been partly funded by EU contract IST-FET IQ FP6-516169, and ANR Bingo2 (MDCO 2007-2010). Most of this research has been done while J. Besson was affiliated to INSA Lyon.

References

1. Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., Verkamo, A.: Fast discovery of association rules. In: Advances in Knowledge Discovery and Data Mining, pp. 307–328. AAAI Press (1996)
2. Basu, S., Davidson, I., Wagstaff, K.: Constrained Clustering: Advances in Algorithms, Theory and Applications. Chapman & Hall/CRC Press, Data Mining and Knowledge Discovery Series (2008)
3. Besson, J., Robardet, C., Boulicaut, J.F.: Mining a new fault-tolerant pattern type as an alternative to formal concept discovery. In: ICCS'06: Proc. Int. Conf. on Conceptual Structures, *LNCS*, vol. 4068. Springer (2006)
4. Bistarelli, S., Bonchi, F.: Interestingness is not a dichotomy: Introducing softness in constrained pattern mining. In: PKDD'05: Proc. 9th European Conf. on Principles and Practice of Knowledge Discovery in Databases, *LNCS*, vol. 3721, pp. 22–33. Springer (2005)
5. Bonchi, F., Giannotti, F., Lucchese, C., Orlando, S., Perego, R., Trasarti, R.: A constraint-based querying system for exploratory pattern discovery. Information Systems **34**(1), 3–27 (2009)
6. Bonchi, F., Giannotti, F., Mazzanti, A., Pedreschi, D.: Adaptive constraint pushing in frequent pattern mining. In: PKDD'03: Proc. 7th European Conf. on Principles and Practice of Knowledge Discovery in Databases, *LNCS*, vol. 2838, pp. 47–58. Springer (2003)
7. Bonchi, F., Giannotti, F., Mazzanti, A., Pedreschi, D.: Examiner: Optimized level-wise frequent pattern mining with monotone constraint. In: ICDM 2003: Proc. 3rd International Conf. on Data Mining, pp. 11–18. IEEE Computer Society (2003)
8. Boulicaut, J.F., De Raedt, L., Mannila, H. (eds.): Constraint-Based Mining and Inductive Databases, *LNCS*, vol. 3848. Springer (2005)
9. Bucila, C., Gehrke, J.E., Kifer, D., White, W.: Dualminer: A dual-pruning algorithm for itemsets with constraints. Data Mining and Knowledge Discovery Journal **7**(4), 241–272 (2003)
10. Calders, T., Rigotti, C., Boulicaut, J.F.: A survey on condensed representations for frequent sets. In: Constraint-based Mining and Inductive Databases, *LNCS*, vol. 3848, pp. 64–80. Springer (2005)
11. Cerf, L., Besson, J., Robardet, C., Boulicaut, J.F.: Closed patterns meet n-ary relations. ACM Trans. on Knowledge Discovery from Data **3**(1) (2009)
12. Cheng, H., Yu, P.S., Han, J.: Ac-close: Efficiently mining approximate closed itemsets by core pattern recovery. In: ICDM, pp. 839–844 (2006)
13. De Raedt, L., Guns, T., Nijssen, S.: Constraint programming for itemset mining. In: KDD'08: Proc. 14th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, pp. 204–212 (2008)
14. Ganter, B.: Two basic algorithms in concept analysis. Tech. rep., Germany Darmstadt : Technisch Hochschule Darmstadt, Preprint 831 (1984)
15. Ganter, B., Stumme, G., Wille, R.: Formal Concept Analysis, Foundations and Applications, *LNCS*, vol. 3626. Springer (2005)
16. Goethals, B., Zaki, M.J. (eds.): Frequent Itemset Mining Implementations, vol. 90. CEUR-WS.org, Melbourne, Florida, USA (2003)

17. Han, J., Pei, J., Yin, Y., Mao, R.: Mining frequent patterns without candidate generation: A frequent-pattern tree approach. Data Mining and Knowledge Discovery **8**(1), 53–87 (2004)
18. Hand, D.J., Adams, N.M., Bolton, R.J. (eds.): Pattern Detection and Discovery, ESF Exploratory Workshop Proceedings, *LNCS*, vol. 2447. Springer (2002)
19. Kifer, D., Gehrke, J.E., Bucila, C., White, W.M.: How to quickly find a witness. In: Constraint-Based Mining and Inductive Databases, pp. 216–242 (2004)
20. Liu, J., Paulsen, S., Sun, X., Wang, W., Nobel, A.B., Prins, J.: Mining approximate frequent itemsets in the presence of noise: Algorithm and analysis. In: SDM (2006)
21. Mannila, H., Toivonen, H.: Levelwise search and borders of theories in knowledge discovery. In: Data Mining and Knowledge Discovery journal, vol. 1(3), pp. 241–258. Kluwer Academic Publishers (1997)
22. Morik, K., Boulicaut, J.F., Siebes, A. (eds.): Local Pattern Detection, International Dagstuhl Seminar Revised Selected Papers, *LNCS*, vol. 3539. Springer (2005)
23. Pasquier, N., Bastide, Y., Taouil, R., Lakhal, L.: Efficient mining of association rules using closed itemset lattices. Information Systems **24**(1), 25–46 (1999)
24. Pei, J., Tung, A.K.H., Han, J.: Fault-tolerant frequent pattern mining: Problems and challenges. In: DMKD. Workshop (2001)
25. Soulet, A., Crémilleux, B.: An efficient framework for mining flexible constraints. In: PaKDD'05: Pacific-Asia Conf. on Knowledge Discovery and Data Mining, *LNCS*, vol. 3518, pp. 661–671. Springer (2005)
26. Uno, T., Kiyomi, M., Arimura, H.: Lcm ver. 2: Efficient mining algorithms for frequent/closed/maximal itemsets. In: FIMI'04, Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations, *CEUR Workshop Proceedings*, vol. 126. CEUR-WS.org (2004)
27. Uno, T., Kiyomi, M., Arimura, H.: LCM ver.3: collaboration of array, bitmap and prefix tree for frequent itemset mining. In: OSDM'05: Proc. 1st Int. Workshop on Open Source Data Mining, pp. 77–86. ACM Press (2005)
28. Yang, C., Fayyad, U., Bradley, P.S.: Efficient discovery of error-tolerant frequent itemsets in high dimensions. In: SIGKDD, pp. 194–203. ACM Press, San Francisco, California, USA (2001)
29. Zaki, M.J.: Scalable algorithms for association mining. IEEE Trans. Knowl. Data Eng. **12**(3), 372–390 (2000)
30. Zhang, M., Wang, W., Liu, J.: Mining approximate order preserving clusters in the presence of noise. In: ICDE, pp. 160–168 (2008)