Constraint Programming for Itemset Mining

Luc De Raedt Tias Guns Siegfried Nijssen

Katholieke Universiteit Leuven, Department of Computer Science
Celestijnenlaan 200A, Leuven, Belgium
{luc.deraedt,tias.guns,siegfried.nijssen}@cs.kuleuven.be

ABSTRACT

The relationship between constraint-based mining and constraint programming is explored by showing how the typical constraints used in pattern mining can be formulated for use in constraint programming environments. The resulting framework is surprisingly flexible and allows us to combine a wide range of mining constraints in different ways. We implement this approach in off-the-shelf constraint programming systems and evaluate it empirically. The results show that the approach is not only very expressive, but also works well on complex benchmark problems.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database applications—*Data Mining*; F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic—*Logic and Constraint Programming*

General Terms

Algorithms, Theory

Keywords

Itemset Mining, Constraint Programming

1. INTRODUCTION

For quite some time, the data mining community has been interested in *constraint-based mining*, that is, the use of constraints to specify the desired properties of the patterns to be mined [1, 4, 5, 6, 8, 11, 12, 15, 16, 10]. The task of the data mining system is then to generate all patterns satisfying the constraints. A wide variety of constraints for local pattern mining exist and have been implemented in an even wider range of specific data mining systems.

On the other hand, the artificial intelligence community has studied several types of constraint-satisfaction problems and contributed many *general purpose* algorithms and systems for solving them. These approaches are now gathered in the area of *constraint programming* [2, 13]. In constraint programming, the user specifies the model, that is, the set of constraints to be satisfied, and the constraint solver generates solutions. Thus, the goals of constraint programming and constraint based mining are similar (not to say identical); it is only that constraint programming targets *any* type of constraint satisfaction problem, whereas constraint-based mining *specifically* targets data mining applications. Therefore, it is surprising that despite the similarities between these two endeavours, the two fields have evolved independently of one another, and also, that – to the best of the authors' knowledge – constraint programming tools and techniques have not yet been applied to pattern mining, and, vice versa, that ideas and challenges from constraint-based mining have not yet been taken up by the constraint programming community.

In this paper, we bridge the gap between these two fields by investigating how standard constraint-programming techniques can be applied to a wide range of pattern mining problems. To this aim, we first formalize most well-known constraint-based mining problems in terms of constraint programming terminology. This includes constraints such as frequency, closedness and maximality, and constraints that are monotonic, anti-monotonic and convertible, as well as variations of these constraints, such as δ-closedness. We then incorporate them in off-the-shelf and state-of-the-art constraint programming tools, such as Gecode[1] [14] and ECLiPSe[2] [2], and run experiments. The results are surprising in that 1) using the constraint programming approach, it is natural to combine complex constraints in a flexible manner (for instance, $\delta-$closedness in combination with monotonic and anti-monotonic constraints); unlike in the existing constraint-based mining systems, this *does not* require modifications to the underlying solvers; 2) the search strategy of constraint programming systems turns out to parallel the search strategy of existing, specialized constraint-based mining approaches among which Eclat [16], LCM [15] and DualMiner [5]; 3) even though the constraint programming methods were not meant to cope with the specifics of data mining (such as coping with large data sets, having 10 000s of constraints to solve), and even though the focus of this study is not on the development of efficient algorithms, it turns out that existing constraint programming systems already perform quite well as compared to dedicated data mining solvers in that on a number of benchmark problems their performance is similar, and in some cases even better com-

[1] `http://www.gecode.org/`

[2] `http://eclipse.crosscoreop.com/`

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
KDD'08, August 24–27, 2008, Las Vegas, Nevada, USA.
Copyright 2008 ACM 978-1-60558-193-4/08/08 ...$5.00.

pared to state-of-the-art itemset miners [10]. At the same time, it should be clear that – in principle – the resulting constraint programming methods can be further optimized towards data mining.

This paper is organized as follows: in Section 2 we introduce a wide variety of constraints for itemset mining problems; in Section 3 we introduce the main principles of constraint programming systems; Section 4 then shows how the contraint-based mining problems can be formulated using constraint programming principles; Section 5 then compares the operation of constraint programming systems with those of dedicated itemset mining algorithms. Section 6 reports on an experimental evaluation comparing an off-the-shelf constraint programming system with state-of-the-art itemset mining implementations, and finally, Section 7 concludes.

2. ITEMSET MINING

Let $\mathcal{I} = \{1, \dots, m\}$ be a set of items, and $\mathcal{T} = \{1, \dots, n\}$ a set of transactions. Then an itemset database $\mathcal{D}$ is a binary matrix of size $n \times m$. Furthermore, $\varphi : 2^{\mathcal{I}} \to 2^{\mathcal{T}}$ is a function that maps an itemset I to the set of transactions from $\mathcal{T}$ in which all its items occur, that is,

$$\varphi(I) = \{t \in \mathcal{T} | \forall i \in I : \mathcal{D}_{ti} = 1\}$$

Dually, $\psi : 2^{\mathcal{T}} \to 2^{\mathcal{I}}$ is a function that maps a transaction-set T to the set of all items from $\mathcal{I}$ shared by all transactions in T, that is,

$$\psi(T) = \{i \in \mathcal{I} | \forall t \in T : \mathcal{D}_{ti} = 1\}$$

It is well-known that φ and ψ define a Galois connection between the lattices $(\mathcal{T}, \subseteq)$ and $(\mathcal{I}, \subseteq)$. This means that the following properties are satisfied:

$$\begin{aligned} \forall T_1, T_2 \subseteq \mathcal{T} &: T_1 \subseteq T_2 \to \psi(T_2) \subseteq \psi(T_1) \\ \forall I_1, I_2 \subseteq \mathcal{I} &: I_1 \subseteq I_2 \to \varphi(I_2) \subseteq \varphi(I_1) \\ \forall I \subseteq \mathcal{I} &: \varphi(I) = \cap_{i \in I} \varphi(\{i\}) \\ \forall T \subseteq \mathcal{T} &: \psi(T) = \cap_{t \in T} \psi(\{t\}) \end{aligned}$$

In the remainder of this section, we will introduce several well-known constraints in itemset mining by making use of these operators. We will show that many itemset mining problems can be formulated as a search for pairs (I, T), where I is an itemset and T a transaction set.

Frequent Itemsets. Our first example are the traditional frequent itemsets [1]. The search for these itemsets can be seen as searching for solution pairs (I, T), such that

$$T = \varphi(I) \quad (1)$$
$$|T| \geq \theta \quad (2)$$

where θ is a frequency threshold. The first constraint specifies that T must equal the set of transactions in which I occurs; the next constraint is the well-known minimum frequency requirement: the absolute number of transactions in which I occurs must be at least θ. The properties of the φ operator imply that minimum frequency is an *anti-monotonic* constraint: every subset of an itemset that satisfies the frequency constraint, also satisfies the constraint.

Anti-Monotonic Constraints. Other examples of anti-monotonic constraints are maximum itemset size and maximum total itemset cost [5, 4]. Assume that every item has a cost c_i (in the case of a size constraint, $c_i = 1$). Then we satisfy a maximum cost (size) constraint, for $c(I) = \sum_{i \in I} c_i$, if

$$c(I) \leq \gamma. \quad (3)$$

Monotonic Constraints. The duals of anti-monotonic constraints are monotonic constraints. Maximum frequency, minimum size and minimum cost are examples of monotonic constraints [5, 4]. Maximum frequency can be formulated similar to (2) as:

$$|T| \leq \theta \quad (4)$$

while minimum cost is expressed similar to (3) by

$$c(I) \geq \gamma. \quad (5)$$

These constraints are called *monotonic* as every superset of an itemset that satisfies the constraints, also satisfies these constraints.

Convertible Anti-Monotonic Constraints. Some constraints are neither monotonic nor anti-monotonic, but still have properties that can be exploited in mining algorithms. One such class of constraints are the convertible (anti-)monotonic constraints [12]. Let us illustrate these by the minimum average cost constraint, which can be specified as

$$c(I)/|I| \geq \gamma. \quad (6)$$

This constraint is called convertible anti-monotone as we can compute an order on the items in $\mathcal{I}$ such that for any itemset $I \subseteq \mathcal{I}$, every prefix I' of the items in I sorted in this order, also satisfies the constraint. In this case, we can order the items decreasing in cost: the average cost can only go up if we remove the item with the lowest cost.

Closed Itemsets. Closed Itemsets are a popular *condensed* representation for the set of all frequent itemsets and their frequencies [15]. Itemsets are called closed when they satisfy

$$I = \psi(T) \quad (7)$$

in addition to constraint (1); alternatively this can be formulated as $I = \psi(\varphi(I))$. A generalization are the δ-closed itemsets [7], which are itemsets that satisfy

$$\forall I' \supset I : |\varphi(I')| < (1 - \delta)|T|; \quad (8)$$

the traditional closed itemsets are a special case with $\delta = 0$.

Maximal Itemsets. *Maximal* frequent itemsets are another condensed representation for the set of frequent itemsets [6]. In addition to the frequent itemset constraints (2) and (1) these itemsets also satisfy

$$\forall I' \supset I : |\varphi(I')| < \theta; \quad (9)$$

all itemsets that are a superset of a maximal itemset are infrequent, while all itemsets that are subsets are frequent. Maximal frequent itemsets constitute a *border* between itemsets that are frequent and not frequent.

Emerging Patterns. If two databases are given, one can be interested in finding itemsets that distinguish these two databases. In other words, one is interested in finding triples $(I, T^{(1)}, T^{(2)})$, containing an itemset I and two transaction sets $T^{(1)}$ and $T^{(2)}$, such that some distinguishing property

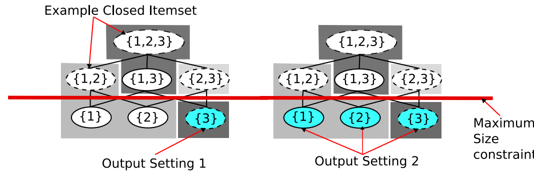

Figure 1: There are two combinations of closedness and maximum size

between $T^{(1)}$ and $T^{(2)}$ holds. Among the many ways to score a pattern's ability to distinguish two datasets is the following constraint:

$$\begin{aligned} T^{(1)} &= \varphi_1(I) \\ T^{(2)} &= \varphi_2(I) \\ |T^{(1)}|/|T^{(2)}| &\geq \rho, \end{aligned}$$

for a given threshold ρ; we assume that for every database we have separate φ and ψ operators. An itemset that satisfies these constraints is called *emerging* [8].

COMBINING CONSTRAINTS. As pointed out in the introduction, it can also be interesting to combine constraints. Defining combinations is not always straightforward when working with constraints such as maximality and closedness [3]. For example, assume we want to mine for $\delta-$closed frequent itemsets that have a size lower than a threshold. Two interpretations are possible. We can define closedness with respect to the set of all frequent itemsets, which means we combine (1), (2), (3) and (8) into:

$$\begin{aligned} T &= \varphi(I) \\ |T| &\geq \theta \\ \forall I' \supset I : |\varphi(I')| &< (1-\delta)|T| \\ c(I) &\leq \gamma \end{aligned}$$

The other interpretation is that we mine for the itemsets that are $\delta-$closed within the set of small itemsets:

$$\begin{aligned} T &= \varphi(I) \\ |T| &\geq \theta \\ \forall I' \supset I : \big(|\varphi(I')| < (1-\delta)|T| &\vee \; c(I') > \gamma\big) \\ c(I) &\leq \gamma \end{aligned}$$

The difference between these two settings is illustrated in Figure 1 for $\delta = 0$ and $\gamma = 1$. Itemsets closed according to constraint (8) are dashed in this figure. In the first setting, only the itemset $\{3\}$ satisfies the constraints; in the second setting, the itemsets $\{1\}$ and $\{2\}$ are also closed considering the maximum size constraint.

Similarly, combinations of maximality and anti-monotonic constraints also have 2 interpretations. Further combinations can be obtained by combining them with emerging patterns. The challenge that we address in this paper, is how to solve such a broad range of queries and their combinations in a unified framework.

3. CONSTRAINT PROGRAMMING

Constraint programming is a declarative programming paradigm: instead of specifying how to solve a problem, the user only has to specify the problem itself. The constraint programming system is then responsible for solving it. Constraint programming systems solve constraint satisfaction problems (CSP). A CSP $\mathcal{P} = (\mathcal{V}, D, \mathcal{C})$ is specified by

- a finite set of variables $\mathcal{V}$;
- an initial domain D, which maps every variable $v \in \mathcal{V}$ to a finite set of integers $D(v)$;
- a finite set of constraints $\mathcal{C}$.

Algorithm 1 Constraint-Search(D)

1: D :=propagate(D)
2: **if** D is a false domain **then**
3:     **return**
4: **end if**
5: **if** $\exists x \in \mathcal{V} : |D(x)| > 1$ **then**
6:     $x := \arg\min_{x \in \mathcal{V}, D(x) > 1} f(x)$
7:     **for all** $d \in D(x)$ **do**
8:         Constraint-Search($D \cup \{x \mapsto \{d\}\}$)
9:     **end for**
10: **else**
11:     Output solution
12: **end if**

A *constraint* $C(x_1, \ldots, x_k) \in \mathcal{C}$ is a boolean function from variables $\{x_1, \ldots, x_k\} \subseteq \mathcal{V}$. A constraint is called *unary* if it involves one variable and *binary* if it involves two. A domain D' is called *stronger* than the initial domain D if $D'(x) \subseteq D(x)$ for all $x \in \mathcal{V}$. A domain is *false* if there exists an $x \in \mathcal{V}$ such that $D(x) = \emptyset$; a variable $x \in \mathcal{V}$ is called *fixed* if $|D(x)| = 1$. A solution to a CSP is a domain D' that fixes all variables ($\forall x \in \mathcal{V} : |D'(x)| = 1$) and satsifies all constraints: abusing notation, we must have that $\forall C(x_1, \ldots, x_k) \in \mathcal{C} : C(D'(x_1), \ldots, D'(x_k)) = 1$; furthermore D' must be stronger than D, which guarantees that every variable has a value from its initial domain $D(x)$.

EXAMPLE 1. *Assume we have four people that we want to allocate to 2 offices, and that every person has a list of other people that he does not want to share an office with. Furthermore, every person has identified rooms he does not want to occupy. We can represent an instance of this problem with four variables, which represent the persons, and inequality constraints, which encode the room-sharing constraints:*

$$D(x_1) = D(x_2) = D(x_3) = D(x_4) = \{1, 2\}$$
$$\mathcal{C} = \{x_1 \neq 2, x_1 \neq x_2, x_3 \neq x_4\}.$$

The simplest algorithm to solve CSPs enumerates all possible fixed domains, and evaluates all constraints on each of these domains; clearly this approach is inefficient. The outline of a general, more efficient Constraint Programming (CP) system is given in Algorithm 1 above [14]. Essentially, a CP system performs a depth-first search; in each node of the search tree the algorithm branches by assigning values to a variable that is unfixed (line 7). It backtracks when a violation of constraints is found (line 2). The search is further optimized by carefully choosing the variable that is fixed next (line 6); a function $f(x)$ ranks variables, for instance, by determining which variable is involved in most constraints.

The main concept used to speed-up the search is constraint propagation (line 1). Propagation reduces the do-

mains of variables such that the domain remains *locally consistent*. One can formally define many types of local consistencies, but we skip these definitions here. In general, in a locally consistent problem a value d does not occur in the domain of a variable x if it can be determined that there is no solution D' in which $D'(x) = \{d\}$. The main motivation for maintaining local consistencies is to ensure that the backtracking search does not unnecessarily branch, thereby significantly speeding up the search.

To maintain local consistencies *propagators* or propagation rules are used. A propagator takes as input a domain and outputs a stronger, locally consistent domain. The propagation rules are derived by the system from the user specified constraints. A *checking propagator* is a propagator that produces a false domain once the original constraint is violated. Most propagators are checking propagators. The repeated application of propagators can lead to increasingly stronger domains. Propagation continues until a *fixed point* is reached in which the domain does not change any more (line 1). There are many different constraint programming systems, which differ in the type of constraints they support and the way they handle these constraints. Most systems assign priorities to constraints to ensure that propagators of lower computational complexity are evaluated first. The main challenge is to manipulate the propagators such that propagation is as cheap as possible.

EXAMPLE 2 (EXAMPLE 1 CONTINUED). *The initial domain of this problem is not consistent: the constraint $x_1 \neq 2$ cannot be satisfied when $D(x_1) = \{2\}$; consequently 2 is removed from $D(x_1)$. Subsequently, the binary constraint $x_1 \neq x_2$ cannot be satisfied while $x_2 = 1$. Therefore value 1 is removed from the domain of x_2. The propagator for the constraint $x_1 \neq x_2$ has the following form:*

if $D(x_1) = \{d\}$ **then delete** d *from* $D(x_2)$.

After applying all propagators in our example, we obtain a fixed point in which $D(x_1) = \{1\}$ and $D(x_2) = \{2\}$, which means persons one and two have been allocated to an office. Two rooms are possible for person 3. The search branches therefore. For each of these branches, the second inequality constraint is propagated; a fixed point is then reached in which every variable is fixed, and a solution is found.

To formulate itemset mining problems as constraint programming models, we only use variables with binary domains, i.e. $D(x) = \{0, 1\}$ for all $x \in \mathcal{V}$. Furthermore, we make extensive use of two types of constraints. The first is a *summation* constraint, whose general form is as follows:

$$\sum_{x \in V} w_x x \geq \theta. \quad (10)$$

In this constraint, $V \subseteq \mathcal{V}$ is a set of variables and w_x is a weight for variable x and can be either positive or negative.

To make clear how this constraint can be propagated, we show a propagator here, such as implemented in most CP systems. Let us use the following notation: $x^{max} = \max_{d \in D(x)} d$ and $x^{min} = \min_{d \in D(x)} d$. Furthermore, $V^+ = \{x \in V | w_x \geq 0\}$ and $V^- = \{x \in V | w_x < 0\}$.

At any point during the search the following constraint must be satisfied in order for Equation (10) to be satisfied:

$$\sum_{x \in V^-} w_x x^{min} + \sum_{x \in V^+} w_x x^{max} \geq \theta.$$

The correctness of this formula follows from the fact that the lefthand side of the equation denotes the highest value that the sum can still achieve.

A checking propagator derived from the constraint for a variable $x' \in V^+$ conceptually has the following effects:

1: **if** $\sum_{x \in V^-} w_x x^{min} + \sum_{x \in V^+} w_x x^{max} \geq \theta$ **then**
2:     **if** $\sum_{x \in V^-} w_x x^{min} + \sum_{x \in V^+ \setminus \{x'\}} w_x x^{max} < \theta$ **then**
3:         $D(x') = \{1\}$
4:     **end if**
5: **else**
6:     $D(x') = \emptyset$
7: **end if**

Only in line 3 an effective domain reduction takes place. A similar propagator can be derived for a variable $x' \in V^-$.

EXAMPLE 3. *Let us illustrate the application of the propagator for the summation constraint on this problem:*

$x_1 + x_2 + x_3 \geq 2,$
$D(x_1) = \{1\}, D(x_2) = \{0, 1\}, D(x_3) = \{0, 1\};$

In this case, we know that at least one of x_2 and x_3 must have the value 1, but we cannot conclude that either one of these variables is certainly zero or one. The propagator does not change any domains. On the other hand, if

$x_1 + x_2 + x_3 \geq 3,$
$D(x_1) = \{1\}, D(x_2) = \{0, 1\}, D(x_3) = \{0, 1\};$

the propagator determines that $D(x_2) = D(x_3) = \{1\}$.

Another special type of constraints that we will use are *reified constraints*:

$$C \leftrightarrow x,$$

where C is a constraint and x is a boolean variable. A reified constraint binds the value of one variable to the evaluation of a constraint. An example of such a constraint is

$$\sum_{x \in V} w_x x \geq \theta \leftrightarrow x', \quad (11)$$

which states that if the weighted sum of certain variables V is higher than θ, variable x' should be true, and vice-versa.

We can decompose a reified constraint in two directions:

$$C \rightarrow x \quad \text{and} \quad C \leftarrow x.$$

We show the propagation that can be performed for both directions, in the special case of Equation (11). For the direction $C \rightarrow x$, the propagation is:

1: **if** $0 \in D(x)$ and $\sum_{x \in V^-} w_x x^{max} + \sum_{x \in V^+} w_x x^{min} \geq \theta$ **then delete** 0 from $D(x)$
2: **if** $D(x) = \{0\}$ **then** apply propagators for $\neg C$

For the reverse direction the propagation is:

1: **if** $1 \in D(x)$ and $\sum_{x \in V^-} w_x x^{min} + \sum_{x \in V^+} w_x x^{max} < \theta$ **then delete** 1 from $D(x)$
2: **if** $D(x) = \{1\}$ **then** apply propagators for C

It is important to note the difference between a summation constraint C and its reified version $x \rightarrow C$. As we can see from the code above, the propagator for $x \rightarrow C$ is not as

expensive to evaluate as the propagator for C as soon as $1 \notin D(x)$.

Even though the $C \leftrightarrow x$ constraint can be expressed in both ECLiPSe and Gecode, we found that the reified implications $C \leftarrow x$ and $C \rightarrow x$ are not available by default; in our implementations (see Section 6) we use additional variables to express one direction of reified constraints.

4. REFORMULATING CONSTRAINTS ON ITEMSETS

We now introduce the models of itemset mining problems that can be provided to constraint programming systems. We choose the following representation of itemsets and transactions. For every transaction we introduce a variable $T_t \in \{0,1\}$, and for every item a variable $I_i \in \{0,1\}$; thus, we can conceive an itemset I as a vector of length m with binary variables; a transaction set T is a vector of length n.

Theorem 1 (Frequent Itemsets). *Frequent itemset mining is expressed by the following constraints:*

$$\forall t \in \mathcal{T} : T_t = 1 \quad \leftrightarrow \quad \sum_{i \in \mathcal{I}} I_i(1 - \mathcal{D}_{ti}) = 0. \tag{12}$$

$$\forall i \in \mathcal{I} : I_i = 1 \quad \rightarrow \quad \sum_{t \in \mathcal{T}} T_t \mathcal{D}_{ti} \geq \theta. \tag{13}$$

Proof. The first constraint (12) is a reformulation of the coverage constraint (1):

$$\begin{aligned} T = \varphi(I) \quad &= \quad \{t \in \mathcal{T} | \forall i \in I : \mathcal{D}_{ti} = 1\} \\ \Longleftrightarrow \forall t \in \mathcal{T} \quad &: \quad t \in T \leftrightarrow \forall i \in I : \mathcal{D}_{ti} = 1 \\ \Longleftrightarrow \forall t \in \mathcal{T} \quad &: \quad t \in T \leftrightarrow \forall i \in I : 1 - \mathcal{D}_{ti} = 0. \\ \Longleftrightarrow \forall t \in \mathcal{T} \quad &: \quad T_t = 1 \leftrightarrow \sum_{i \in \mathcal{I}} I_i(1 - \mathcal{D}_{ti}) = 0. \end{aligned}$$

The second constraint (13) is derived as follows. We can reformulate the frequency constraint as:

$$\sum_{t \in \mathcal{T}} T_t \geq \theta. \tag{14}$$

Together with the coverage constraint, this constraint defines the frequent itemset mining problem. As argued in the previous section, however, reified constraints can sometimes be desirable. To this purpose, we rewrite the frequency constraint further. First, we can observe that $\forall i \in I : |T| = |T \cap \varphi(\{i\})|$, as $T = \varphi(I) \subseteq \varphi(\{i\})$, and therefore that in a valid solution

$$\begin{aligned} \forall i \in I \quad &: \quad |T \cap \varphi(\{i\})| \geq \theta \\ \Longleftrightarrow \forall i \in \mathcal{I} \quad &: \quad I_i = 1 \rightarrow \sum_{t \in \mathcal{T}} T_t \mathcal{D}_{ti} \geq \theta. \end{aligned}$$

Please note that reification increases the number of constraints significantly; it will depend on the problem setting if this increase is still beneficial in the end. In the next section we will study how a constraint programming system operates in practice on these constraints. □

Many other anti-monotonic constraints can also be specified in a straightforward way.

Theorem 2 (Anti-Monotonic Constraints). *The maximum total cost constraint is expressed by:*

$$\sum_{i \in \mathcal{I}} c_i I_i \leq \gamma.$$

Theorem 3 (Monotonic Constraints). *A monotonic minimum cost constraint is specified by*

$$\sum_{i \in \mathcal{I}} c_i I_i \geq \gamma$$

or, equivalently, reified as

$$\forall t \in \mathcal{T} : T_t = 1 \quad \rightarrow \quad \sum_{i \in \mathcal{I}} c_i I_i \mathcal{D}_{ti} \geq \gamma. \tag{15}$$

Proof. The reified version of the constraint exploits that $I \subseteq \psi(\{t\})$ for every $t \in \varphi(I)$; starting from (5):

$$\begin{aligned} \sum_{i \in I} c_i \geq \gamma \Longleftrightarrow \forall t \in T \quad &: \quad \sum_{i \in (I \cap \psi(\{t\}))} c_i \geq \gamma \\ \Longleftrightarrow \forall t \in \mathcal{T} \quad &: \quad T_t = 1 \rightarrow \sum_{i \in \mathcal{I}} c_i I_i \mathcal{D}_{ti} \geq \gamma. \end{aligned}$$

□

Theorem 4 (Convertible Constraints). *The convertible minimum average cost constraint is specified as follows:*

$$\sum_{i \in \mathcal{I}} (c_i - \gamma) I_i \geq 0$$

Equivalently, a reified version can be used similar to (15).

Proof. This follows from rewriting constraint (6):

$$\sum_{i \in I} c_i / |I| \geq \gamma \Leftrightarrow \sum_{i \in \mathcal{I}} c_i I_i \geq \gamma \sum_{i \in \mathcal{I}} I_i \Leftrightarrow \sum_{i \in \mathcal{I}} (c_i - \gamma) I_i \geq 0.$$

□

Theorem 5 (Closed Itemsets). *The frequent closed itemset mining problem is specified by the conjunction of the coverage constraint* (12)*, the frequency constraint* (13) *and*

$$\forall i \in \mathcal{I} : I_i = 1 \leftrightarrow \sum_{t \in \mathcal{T}} T_t(1 - \mathcal{D}_{ti}) = 0. \tag{16}$$

The more general $\delta-$closed itemsets are specified by

$$\forall i \in \mathcal{I} : I_i = 1 \leftrightarrow \sum_{t \in \mathcal{T}} T_t(1 - \delta - \mathcal{D}_{ti}) \leq 0. \tag{17}$$

Proof. Formulation (16) follows from the second Galois operator in (7) similar to the reformulation of (12) above. We skip the derivation for the δ-closed itemsets due to lack of space. □

Theorem 6 (Maximal Frequent Itemset Mining). *The maximal frequent itemset mining problem is specified by the coverage constraint* (12) *and constraint*

$$\forall i \in \mathcal{I} : I_i = 1 \quad \leftrightarrow \quad \sum_{t \in \mathcal{T}} T_t \mathcal{D}_{ti} \geq \theta \tag{18}$$

Proof. The maximality constraint is reformulated as follows:

$$\begin{aligned} \forall I' \supset I \quad &: \quad |\varphi(I')| < \theta \\ \Longleftrightarrow \forall i \in \mathcal{I} - I \quad &: \quad |\varphi(I \cup \{i\})| < \theta \\ \Longleftrightarrow \forall i \in \mathcal{I} - I \quad &: \quad |T \cap \varphi(\{i\})| < \theta \\ \Longleftrightarrow \forall i \in \mathcal{I} \quad &: \quad I_i = 0 \rightarrow \sum_{t \in \mathcal{T}} T_t \mathcal{D}_{ti} < \theta \end{aligned} \tag{19}$$

Together with the minimum frequency constraint (13) this becomes a two sided reified constraint. □

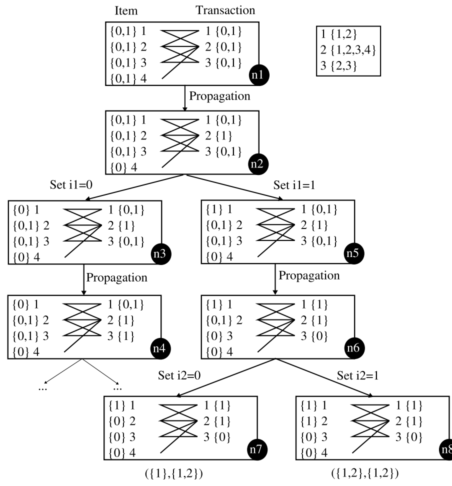

Figure 2: Search tree for the frequent itemset mining problem.

THEOREM 7 (EMERGING PATTERNS). *The problem of finding frequent emerging patterns is specified by:*

$$\forall k \in \{1,2\} : \forall t \in \mathcal{T}^{(k)} : T^{(k)}t = 1 \quad \leftrightarrow \quad \sum_{i \in \mathcal{I}} I_i(1 - \mathcal{D}^{(k)}_{ti}) = 0.$$

$$\forall i \in \mathcal{I} : I_i = 1 \quad \rightarrow \quad \sum_{t \in \mathcal{T}_1} T_t \mathcal{D}^{(1)}_{ti} \geq \theta.$$

$$\forall i \in \mathcal{I} : I_i = 1 \quad \rightarrow \quad \sum_{t \in \mathcal{T}^{(1)}} T^{(1)}_t \mathcal{D}^{(1)}_{ti} - \rho \sum_{t \in \mathcal{T}^{(2)}} T^{(2)}_t \mathcal{D}^{(2)}_{ti} \geq 0,$$

PROOF. This follows from the reification of

$$|T^{(1)}|/|T^{(2)}| \geq \rho \Leftrightarrow \sum_{t \in \mathcal{T}^{(1)}} T^{(1)}_t - \rho \sum_{t \in \mathcal{T}^{(2)}} T^{(2)}_t \geq 0;$$

furthermore, given that two datasets are given, coverage is expressed for both datasets. □

5. CONSTRAINT PROGRAMMING SYSTEMS AS ITEMSET MINERS

We now investigate the behavior of constraint programming systems applied to itemset mining problems and compare this to standard constraint-based mining techniques.

Let us start with the *frequent itemset* mining problem. The search tree for an example database is illustrated in Figure 2. We use a minimum frequency threshold of $\theta = 2$. In the initial search node (n_1) the propagator for frequency constraint (13) sums for each item the number of transactions having that item, and determines that item 4 is only covered by 1 transaction, and therefore sets $D(I_4) = \{0\}$. The propagators for coverage constraint (12) determine that transaction 2 is covered by all items, and hence $D(T_2) = \{1\}$. This leads to the domain in node n_2, where we have to branch. One of these branches leads to node n_5, setting $D(I_1) = \{1\}$, thus including item 1 in the itemset. Coverage propagation sets transaction $D(T_3) = \{0\}$; frequency propagation determines that itemset $\{1,3\}$ is not frequent and sets $D(I_3) = \{0\}$. Coverage propagation determines that transaction 1 is covered by all remaining items and sets $D(T_1) = \{1\}$. Propagation stops in node n_6. Here both possibilities in $D(I_2)$ are considered, but no further propagation is possible and we find the two frequent itemsets $\{1\}$ and $\{1,2\}$.

In this example, the reified constraint is responsible for setting an item $D(I_i) = \{0\}$. Without reification the frequency constraint would never influence the domain of items. In our example, the system would continue branching below node n_7 to set $D(I_3) = \{1\}$, and only then find out that the resulting domain is false. By using the reified constraints, the system remembers which items were infrequent earlier, and will not try to add these deeper down the search tree. This example illustrates a key point: by using reified constraints, a CP system behaves very similar to other well-known depth-first itemset miners such as Eclat [16] and FP-Growth [11]. For a given itemset also these miners maintain which items can still be added to yield a frequent itemset (in FP-Growth, for instance, a projected database only contains such items). The transaction variables store a transaction set in a similar way as Eclat does: the variables that still have $1 \in D(T_t)$ represent transactions that are covered by an itemset during the search. A difference between well-known itemset miners and a CP system is that a CP system also maintains a set of transactions that are fixed to 1; furthermore, the CP system explicitly sets item variables to zero, while other systems usually do this implicitly by skipping them.

Let us consider the *maximal frequent itemset* mining problem next. Compared to the search tree of Figure 2, the search tree for maximal frequent itemset mining is different below node n_6: applying the additional propagator for the maximality constraint (19), the CP system would now conclude that a sufficient number of transactions containing item 2 have been fixed to 1 and would remove 0 from the domain of item I_2. As all variables are fixed, the search stops here and itemset $\{1,2\}$ is found. This behavior is very similar to that of a well-known maximal frequent itemset miner: MAFIA [6] (and its generalization DualMiner [5]) uses the set of 'unavoidable' transactions (obtained by computing the supporting transactions of the Head-Union-Tail (HUT) itemset), and immediately adds all items if the HUT turns out to be frequent.

Likewise, we can consider *closed frequent itemset mining*: in node n_6 the CP system would conclude that all transactions with $D(T_t) = \{1\}$ support item 2, and would remove value 0 from $D(I_2)$ due to constraint (16); in general, this means that for every itemset, the items in its closure are computed; if an item is in the closure, but is already fixed to 0, the search backtracks, otherwise, the search continues with all items added. The same search strategy is employed by the well-known itemset miner LCM [15].

If we look at a *monotonic* minimum cost constraint, where we assume $c_1 = c_2 = c_4 = 1$ and $c_3 = 2$, and the cost threshold is at $\gamma = 3$, the search tree would differ already at node n_2: if the reified constraint (15) is used, the items in transaction 1 are not expensive enough to exceed the cost threshold, and $D(T_1) = \{0\}$ is set. The effect is that item 1 does not have sufficient support any more, and is set to zero. This kind of pruning for monotonic constraints was called ExAnte pruning and was implemented in the ExaMiner [4].

Until now we did not specify how a CP system selects its next variable to fix (Algorithm 1, line 6). A careful choice can influence the efficiency of the search. This situation occurs when dealing with *convertible anti-monotonic*

	LCM [15]	MAFIA [6]	ExAMiner [4]	DualMiner [5]	CP
Constraints on data					
Minimum frequency	X	X	X	X	X
Maximum frequency				X	X
Emerging patterns					X
Condensed Representations					
Maximal	X	X		X	X
Closed	X	X			X
$\delta-$Closed					X
Constraints on syntax					
Max/Min total cost			X	X	X
Minimum average cost			X		X
Max/Min size	X	X	X	X	X

Table 1: Comparison of Itemset Miners

constraints such as minimum average cost constraints. We can show that for a well-chosen order of variables, which reflects the cost constraint, the CP system will never encounter a false domain (similar to systems such as FP-Growth extended with convertible constraints [12]).

Summarizing our observations, it turns out that the search strategy employed by many itemset miners parallels that of standard CP systems applied to constraint-based mining problems. Furthermore, the CP approach is able to deal with *combinations* of constraints in a straight-forward manner. A comparison is given in Table 1[3].

6. IMPLEMENTATION AND EXPERIMENTS

In this section we study the practical benefits and drawbacks of using state-of-the-art CP systems. In the first section, we consider this issue from a modeling perspective: how involved is it to specify an itemset mining task in CP systems? In the second section, we study the performance perspective, answering the question: how efficient are CP systems compared to existing itemset mining systems?

6.1 Modeling Efficiency

To illustrate how easy it is to specify itemset mining problems, we develop a model for standard frequent itemset mining in the ECLiPSe Constraint Programming System, a logic programming based solver with a declarative modeling language [2]. This model is given in Algorithm 2. It is almost identical to the formal notation.

This model generates all frequent itemsets, given a 2-dimensional array D, minimum frequency $Freq$, and predicate *prodlist(In1,In2,Out)*, which returns a list where each element is the multiplication of the corresponding elements in the input lists. Next to procedures for reading data all functionality is available by default in ECLiPSe.

Models for other constraints, and combinations of them, can be created in a similar way. For example, if we only want maximal itemsets, we replace line 10 by `I #= (sum(PList) #>= Freq)`. If we only want itemsets of size at least 2, then we can add before line 6: `sum(Items) #>= 2`.

A similar model can be specified using the Gecode CP library [14]. Compared to the development of specialized algorithms, significantly less effort is needed to model the problem in CP systems

[3]These results are based on the parameters of the most recent implementations of the original authors, or, if not publicly available, on the original paper of these authors.

Algorithm 2 Frequent Itemset Mining in ECLiPSe

```
   % Input: D: the data matrix; Freq: frequency threshold
   % Output: Items: an itemset; Trans: a transaction set
1. fim_clp(D, Freq, Items, Trans) :-
2.   dim(D, [NrI,NrT]),
3.   length(Items, NrI),            % decision variables
4.   length(Trans, NrT),
5.   Items :: [0..1], Trans :: [0..1],
6.   (  foreach(I,Items), count(K,1,_),        % model
7.      param(Trans, NrT, D, Freq)
8.   do Col is D[1..NrT,K],
9.      prodlist(Trans, Col, PList),
10.     I => (sum(PList) #>= Freq)
                 % ∀i ∈ I : I_i = 1 → Σ_{t∈T} T_t D_ti ≥ θ
11.  ),
12.  (  foreach(T,Trans), count(K,1,_),
13.     param(Items, NrI, D)
14.  do Row is D[K,1..NrI],
15.     prodlist_compl(Items, Row, PList_c),
16.     T #= (sum(PList_c) #= 0)
                 % ∀t ∈ T : T_t = 1 ↔ Σ_{i∈I} I_i(1 − D_ti) = 0
17.  ),
18.  labeling(Items).
```

6.2 Computational Efficiency

In these experiments we only use the Gecode constraint programming system [14]; we found that ECLiPSe was unable to handle the amounts of constraints that are needed to cope with larger datasets. We implemented models for frequent itemset mining, closed itemset mining, maximal itemset mining, and combinations of frequency, cost and size constraints. We compare with the most recent implementations of LCM and Mafia from the FIMI repository [10]; furthermore, we obtained the PATTERNIST system, which implements the ExAnte property [4]. All these systems are among the most efficient systems available. We use data from the UCI repository[4]. Properties of the data are listed in Table 2. To deal with missing values we preprocessed each dataset in the same way as [9]: we first eliminated all attributes having more than 10% of missing values and then removed all examples (transactions) for which the remaining attributes still had missing values. Numerical attributes were binarized by using unsupervised discretization with 4 bins. Where applicable, we generated costs per items randomly using a uniform distribution between 0 and 200. Experiments were run on PCs with Intel Core 2 Duo E6600 processors and 4GB of RAM, running Ubuntu Linux. The code of our implementation and the datasets used are available on our website[5].

The density of a dataset is calculated by dividing the total number of 1s in the binary matrix by the size of this matrix. We encountered scalability problems for the large datasets in the FIMI challenge, and decided to restrict ourselves to smaller, but dense UCI datasets. We restrict ourselves to dense datasets as these are usually more difficult to mine. The numbers of frequent itemsets in Table 2 for a support threshold of 1% are given as an indication, and were computed using LCM. All our experiments were timed out after 30 minutes. The experiments indicate that additional constraints are needed to mine itemsets for low support values on the Segment data.

[4]`http://archive.ics.uci.edu/ml/`
[5]`http://www.cs.kuleuven.be/~dtai/CP4IM/`

	#Trans.	#Items	Density	#Patterns 1%
German Credit	1000	77	0.28	29 088 485
Letter	20000	74	0.33	1 037 221 530
Segment	2310	74	0.51	(time out)

Table 2: Description of the datasets

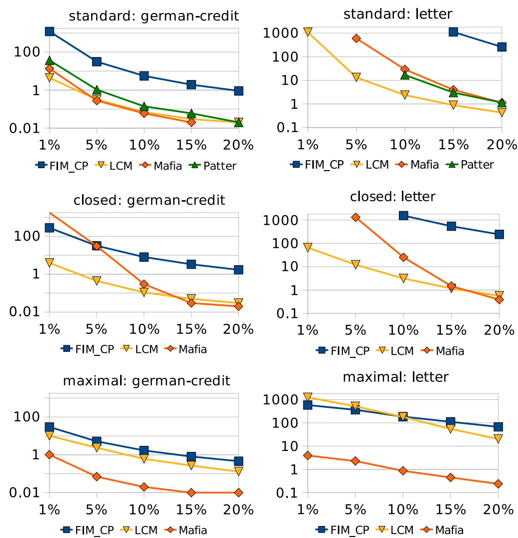

Figure 3: Runtimes of itemset miners on standard problems for different values of minimum support

Experiment 1: Standard Itemset Mining.
In our first experiment, we evaluate how the runtimes of our Gecode-based solver, denoted by *FIM_CP*, compare with those of state-of-the-art itemset miners, for the problems of frequent, maximal and closed itemset mining.

Results for two datasets are given in Figure 3. The experiments show that in most of the standard settings, which require the computation of large numbers of itemsets, FIM_CP is at the moment not very competitive. On the other hand, the experiments also show that the CP solver propagates all constraints as expected; for instance, maximal itemset mining is more efficient than frequent itemset mining. The system behaves very similar to other (specialized) systems from this perspective. In particular, if we compare FIM_CP with LCM, we see that FIM_CP sometimes performs better than LCM as a maximal frequent itemset miner for low support thresholds; similarly FIM_CP sometimes outperforms Mafia as closed itemset miner. This is an indication that further improvements in the implementations of CP solvers could lead to systems that perform satisfactory for most users.

Experiment 2: Standard Constraint-Based Mining.
In this experiment we determine how FIM_CP compares with other systems when additional constraints are employed.

Results for two settings are given in Figure 4. In the first experiment we employed a (monotonic) minimum size constraint in addition to a minimum frequency constraint; in the second a (convertible) maximum average cost constraint. The results are positive: even though for small minimum size constraints the brute force mining algorithms, such as LCM, outperform FIM_CP, FIM_CP does search very effectively when this constraint selects a small number of very

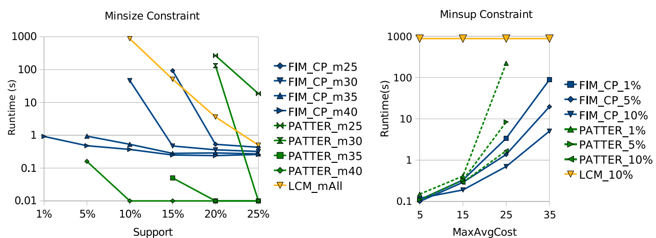

Figure 4: Runtimes of itemset miners on Segment data under constraints

large itemsets (30 items or more); in extreme cases FIM_CP finishes within seconds while other algorithms do not finish within our cut-off time of 30 minutes. PATTERNIST was unable to finish some of these experiments due to memory problems. This indicates that FIM_CP is a competitive solver when the constraints require the discovery of a small number of very large itemsets. The results for convertible constraint are particularly interesting, as we did not optimize the item order in any of our experiments.

Experiment 3: Novel Constraint-Based Mining.
In our final experiments we explore the effects of combining several types of constraints.

Our problem setting is related to the problem of finding itemsets that are predictive for one partition (or class) in the data; one way to find such itemsets is to look for itemsets that have high support in this partition and have low support in the remaining examples. Furthermore, it is desirable to condense the itemsets found; we investigate the use of δ-closedness; δ-closedness has not been studied in combination with other constraints in the literature. Finally, to reduce the complexity of the individual patterns found, we investigate the use of size constraints on the itemsets; we consider both minimum and maximum size constraints.

In our experiments, we divided the Segment dataset randomly in 2 partitions. As default parameters for the constraints we chose: a minimum support of 0.5%, a maximum support of 0.5%, a minimum size of 14, a maximum size of 16 and $\delta = 0.20$. Results are summarized in Table 3.

The issues that we study in this table are the following. First, as discussed in Section 2 and illustrated in Figure 1, there are two ways of combining maximum size and $\delta-$closedness, referred to as Setting 1 and Setting 2. We are interested in the difference in size in the resulting sets of itemsets. In practice it turns out that Setting 1 yields significantly less itemsets than the second setting. Second, we study the influence of the minimum size constraint. The experiments show that the CP system achieves significantly lower runtimes when this additional constraint is applied, thus pushing the constraints effectively, while finding fewer patterns. Third, we study the influence of the $\delta-$parameter. We mined for several values of δ, as can be seen in Figure 5, without size constraints. The results show for higher values of δ less patterns are returned and the algorithm runs faster.

Please note that the runtimes of the CP system are much lower than those obtained by LCM for the same support threshold; the CP approach is more efficient than running LCM and post-processing its results.

Constraint on $\|I\|$	Setting 1 # Patterns	Setting 1 Time (s)	Setting 2 # Patterns	Setting 2 Time (s)
None	27860	30.27	27860	30.27
$\|I\| \leq 16$	27756	30.20	39017	34.27
$14 \leq \|I\|$	4507	14.61	4507	14.72
$14 \leq \|I\| \leq 16$	4403	14.51	15664	19.25

Table 3: Applying size constraints on Segment data

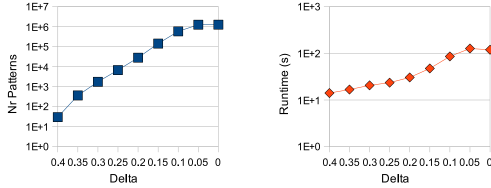

Figure 5: Mining Segment under δ-closedness

7. CONCLUSIONS

We have reformulated itemset mining problems in terms of constraint programming. This has allowed us to implement a novel mining system using standard constraint programming tools. This approach has several benefits. At a conceptual level, constraint programming offers a more uniform, extendible and declarative framework for a wide range of itemset mining problems than state-of-the-art data mining systems. At an algorithmic level, the general purpose constraint programming methods often emulate well-known itemset mining systems. Finally, from an experimental point of view, the results of the constraint programming implementation are encouraging for constraints that select many short itemsets, and are competitive or better for constraints that select few long itemsets. This despite the fact that constraint programming systems are general purpose solvers and were not developed with the large number of constraints needed for data mining in mind.

The main advantage of our method is its generality. The framework can be used to explore new constraints and combinations of constraints much more easily than is currently possible. In our approach, it is no longer necessary to develop new algorithms from scratch to deal with new types of constraints.

There are several open questions for further research. 1) How can constraint-programming algorithms be specialized and optimized for use in data mining? 2) Which other types of constraints can be used for mining with the constraint programming approach? and 3) Can the introduced framework be extended for supporting the mining of structured data, such as sequences, trees and graphs?

To summarize, we have contributed a first step towards bridging the gap between data mining and constraint programming, and have formulated a number of open questions for future research, both on the constraint programming and on the data mining side.

Acknowledgements. Siegfried Nijssen was supported by the EU FET IST project "Inductive Querying", contract number FP6-516169. Tias Guns was supported by the Institute for the Promotion and Innovation through Science and Technology in Flanders (IWT-Vlaanderen). We are grateful to Francesco Bonchi for providing the PATTERNIST system, and to Albrecht Zimmermann and Élisa Fromont for discussions.

8. REFERENCES

[1] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo. Fast discovery of association rules. In *Advances in Knowledge Discovery and Data Mining*, pages 307–328. AAAI Press, 1996.

[2] K. R. Apt and M. Wallace. *Constraint Logic Programming using Eclipse*. Cambridge University Press, New York, NY, USA, 2007.

[3] F. Bonchi and C. Lucchese. On closed constrained frequent pattern mining. In *ICDM*, pages 35–42. IEEE Computer Society, 2004.

[4] F. Bonchi and C. Lucchese. Extending the state-of-the-art of constraint-based pattern discovery. *Data Knowl. Eng.*, 60(2):377–399, 2007.

[5] C. Bucila, J. Gehrke, D. Kifer, and W. M. White. Dualminer: A dual-pruning algorithm for itemsets with constraints. *Data Min. Knowl. Discov.*, 7(3):241–272, 2003.

[6] D. Burdick, M. Calimlim, J. Flannick, J. Gehrke, and T. Yiu. Mafia: A maximal frequent itemset algorithm. *IEEE Trans. Knowl. Data Eng.*, 17(11):1490–1504, 2005.

[7] J. Cheng, Y. Ke, and W. Ng. δ-tolerance closed frequent itemsets. In *ICDM '06: Proceedings of the Sixth International Conference on Data Mining*, pages 139–148, 2006.

[8] G. Dong and J. Li. Efficient mining of emerging patterns: Discovering trends and differences. In *KDD*, pages 43–52, 1999.

[9] E. Frank and I. H. Witten. Using a permutation test for attribute selection in decision trees. In *Proc. 15th International Conf. on Machine Learning*, pages 152–160, 1998.

[10] B. Goethals and M. J. Zaki. Advances in frequent itemset mining implementations: report on FIMI'03. In *SIGKDD Explorations Newsletter*, volume 6, pages 109–117, 2004.

[11] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 1–12, 2000.

[12] J. Pei, J. Han, and L. V. S. Lakshmanan. Mining frequent item sets with convertible constraints. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, pages 433–442, 2001.

[13] F. Rossi, P. van Beek, and T. Walsh. *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. Elsevier Science Inc., 2006.

[14] C. Schulte and P. J. Stuckey. Efficient constraint propagation engines. *Transactions on Programming Languages and Systems*, 2008. To appear.

[15] T. Uno, M. Kiyomi, and H. Arimura. Lcm ver.3: collaboration of array, bitmap and prefix tree for frequent itemset mining. In *OSDM '05: Proceedings of the 1st international workshop on open source data mining*, pages 77–86, 2005.

[16] M. J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. New algorithms for fast discovery of association rules. In *KDD*, pages 283–286, 1997.