Constraint Programming for Mining Borders of Frequent Itemsets
Mohamed-Bachir Belaid, Christian Bessière, Nadjib Lazaar

To cite this version:

HAL Id: lirmm-02310629
https://hal-lirmm.ccsd.cnrs.fr/lirmm-02310629
Submitted on 10 Oct 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Constraint Programming for Mining Borders of Frequent Itemsets

Mohamed-Bachir Belaid, Christian Bessiere and Nadjib Lazaar
LIRMM, University of Montpellier, CNRS, Montpellier, France
{belaid, bessiere, laazaar}@lirmm.fr

Abstract

Frequent itemset mining is one of the most studied tasks in knowledge discovery. It is often reduced to mining the positive border of frequent itemsets, i.e. maximal frequent itemsets. Infrequent itemset mining, on the other hand, can be reduced to mining the negative border, i.e. minimal infrequent itemsets. We propose a generic framework based on constraint programming to mine both borders of frequent itemsets. One can easily decide which border to mine by setting a simple parameter. For this, we introduce two new global constraints, FREQUENTSUBS and INFREQUENTSUPERS, with complete polynomial propagators. We then consider the problem of mining borders with additional constraints. We prove that this problem is coNP-hard, ruling out the hope for the existence of a single CSP solving this problem (unless coNP ⊆ NP).

1 Introduction

Data mining is the art of discovering knowledge from databases. It includes the discovering of frequent itemsets [Agrawal et al., 1994; Han et al., 2000], association rules [Agrawal et al., 1994; Liu et al., 1999], rare itemsets [Szathmary et al., 2007; Adda et al., 2007], sequence patterns [Agrawal and Srikant, 1995], emerging patterns [Dong and Li, 1999], and many other tasks. For many data mining problems, the frequency represents an important metric. Given a frequency threshold s over a transaction dataset, frequent itemsets are those present in at least s transactions. Itemsets present in less than s transactions are called infrequent (or rare). The number of frequent/infrequent itemsets can be huge, making it hard even to print the result. Hence, we often reduce the problem of mining frequent or infrequent itemsets to the problem of mining the borders. The positive border is the set of frequent itemsets with only infrequent supersets, i.e. Maximal Frequent Itemsets (MFIs). The negative border is the set of infrequent itemsets with only frequent subsets, i.e. Minimal Infrequent Itemsets (MIIs). The subsets of the MFIs and the supersets of the MIIs represent the frequent and infrequent itemsets, respectively.

Several techniques have been introduced for mining MFIs [Burdick et al., 2001; Gouda and Zaki, 2001; Uno et al., 2004], or MIIs [Szathmary et al., 2007; Haglin and Manning, 2007; Szathmary et al., 2012]. Despite an obvious duality between MFIs and MIIs, very few papers propose a single framework for mining both MFIs and MIIs. In [Mannila and Toivonen, 1997; Boros et al., 2003; Nourine and Petit, 2012], inferring MFIs from the set of MIIs and vice versa (or also called the dualization) is reduced to computing the minimal transversals of a hypergraph [Berge, 1984]. In fact, MIIs correspond to the minimal transversals of the hypergraph constructed by the complements of the set of MFIs.

In a recent line of work, researchers have taken advantage of the flexibility of constraint programming to model various data mining problems [De Raedt et al., 2008; Khiari et al., 2010; Lazaar et al., 2016; Schaus et al., 2017; Bessiere et al., 2018]. In terms of CPU time, constraint programming-based methods have not yet competed with ad hoc algorithms. However, their flexibility allows the modeling of complex user queries without revising the solving process.

In this paper we propose a generic parameterized model allowing the user to decide which border to mine (i.e., MFIs or MIIs). For this model we need to define two new global constraints: (1) FREQUENTSUBS for mining itemsets having only frequent subsets, and (2) INFREQUENTSUPERS for mining itemsets having only infrequent supersets. We provide a polynomial domain consistency propagator for each of these two constraints. We then address the issue of mining borders in the presence of other constraints. As noticed in [Bonchi and Luccchese, 2004], mining borders under additional constraints can lead to the loss of solutions. This can happen with maximal/minimal borders, but also with closed itemsets or generator itemsets. We prove that it is coNP-hard to find maximal/minimal itemsets among those satisfying a set of additional constraints. This implies that there does not exist any CSP representing the problem of finding maximal/minimal itemsets under additional constraints, unless coNP ⊆ NP. This is an a posteriori justification of the “multi-shot” approaches, such as the one presented in [Negrevergne et al., 2013].

The paper is organized as follows. Section 2 gives some background material. In Section 3 we give a generic model for mining MFIs or MIIs. We define the new global constraints FREQUENTSUBS and INFREQUENTSUPERS and we present their propagators. Section 4 analyzes the problem of mining constrained MFIs or MIIs. We present some empirical results in Section 5. Finally, we conclude in Section 6.
Constraint solvers typically use backtracking search on all constraints. A solution is an assignment where all values belong to the domain of their variable. A solution is an assignment on a set of domains \( \{x_1, \ldots, x_n\} \) such that there does not exist any itemset \( Q \subseteq P \) such that \( \text{freq}(Q) \neq \text{freq}(P) \).

Definition 4 (Generator [Bastide et al., 2000]) A generator is an itemset \( P \) such that there does not exist any itemset \( Q \subseteq P \) such that \( \text{freq}(Q) = \text{freq}(P) \).

Example 2.1 With \( s = 3 \), the dataset in Table 1 has MFIs = \{\( ABE, AC, BCE \)\} and MIIs = \{\( ABC, ACE, D \)\}.

2.2 Constraint Programming (CP)

A CP model specifies a set of variables \( X = \{x_1, \ldots, x_n\} \), a set of domains \( \text{dom} = \{\text{dom}(x_1), \ldots, \text{dom}(x_n)\} \), where \( \text{dom}(x_i) \) is the finite set of possible values for \( x_i \), and a set of constraints \( C \) on \( X \). A constraint \( c_j \in C \) is a relation that specifies the allowed combinations of values for its variables \( \text{var}(c_j) \). An assignment on a set \( Y \subseteq X \) of variables is a mapping from variables in \( Y \) to values, and a valid assignment is an assignment where all values belong to the domain of their variable. A solution is an assignment on \( X \) satisfying all constraints. Constraint programming is the art of writing problems as CP models and solving them by finding solutions. Constraint solvers typically use backtracking search to explore the search space of partial assignments. At each assignment, constraint propagation algorithms (aka, propagators) prune the search space by enforcing local consistency properties such as domain consistency.

A constraint \( c \) on \( \text{var}(c) \) is domain consistent (DC) if and only if, for every \( x_i \in \text{var}(c) \) and every \( d_j \in \text{dom}(x_i) \), there is a valid assignment satisfying \( c \) such that \( x_i = d_j \). Global constraints are constraints defined by a relation on any number of variables. The constraint AllDifferent, specifying that all its variables must take different values is an example of global constraint (see [Rossi et al., 2006]).

Example 2.2 Consider the following instance of a CP model.

\[
X = \{x_1, x_2, x_3\}, \quad \text{dom}(x_1) = \{0, 2\}, \quad \text{dom}(x_2) = \{0, 2, 4\}, \\
\text{dom}(x_3) = \{1, 2, 3, 4\}, \quad \text{and} \quad C = \{x_1 \geq x_2, x_1 + x_2 = x_3\}.
\]

Value 4 for \( x_2 \) will be removed by DC because of constraint \( x_1 \geq x_2 \). Values 1 and 3 for \( x_3 \) will be removed by DC because of constraint \( x_1 + x_2 = x_3 \). This CP model admits the two solutions \( (x_1 = 2, x_2 = 0, x_3 = 2) \) and \( (x_1 = 2, x_2 = 2, x_3 = 4) \).

3 A Generic CP Model for Mining Borders

We present a CP model, \( \text{MODEL}_{D,s,b} \), for mining MFIs or MIIs. \( \text{MODEL}_{D,s,b} \) uses a vector \( x \) of \( n \) Boolean variables, where \( x_i \) represents the presence of the item \( i \) in the itemset. We will use the following notations:

- \( x^{-1}(1) = \{i \in I \mid \text{dom}(x_i) = \{1\}\} \)
- \( x^{-1}(0) = \{i \in I \mid \text{dom}(x_i) = \{0\}\} \)
- \( x^{-1}(\ast) = \{i \in I \mid i \notin x^{-1}(1) \cup x^{-1}(0)\} \)

\[
\text{MODEL}_{D,s,b}(x) = \begin{cases} \text{FREQUENTSUBS}_{D,s}(x) & (1) \\ \text{INFREQUENTSUPERS}_{D,s}(x) & (2) \\ b \iff \text{FREQUENT}_{D,s}(x) & (3) \end{cases}
\]

where (1) is a global constraint that holds if and only if the itemset \( x^{-1}(1) \) has only frequent subsets w.r.t \( s \), (2) is a global constraint that holds if and only if the itemset \( x^{-1}(1) \) has only infrequent supersets w.r.t \( s \) and the Boolean parameter \( b \) reifies the global constraint \( \text{FREQUENT} \) (3). \( \text{FREQUENT}_{D,s} \) holds if and only if \( x^{-1}(1) \) is frequent w.r.t \( s \). We prove that \( \text{MODEL}_{D,s,b} \) is a correct model for mining MFIs or MIIs.

Theorem 1 The set of solutions to \( \text{MODEL}_{D,s,b} \) corresponds to the set of MFIs if \( b = \text{true} \), the set of MIIs otherwise.

Proof. We first prove that MFIs and MIIs always satisfy constraints (1) and (2). All supersets of an MFI are infrequent, thus satisfying (2). Because an MFI is frequent, all its proper subsets are frequent, thus satisfying (1). An MII has only infrequent subsets, thus satisfying (1). Because an MII is infrequent, it has only infrequent supersets, thus satisfying (2). Hence, all MFIs and all MIIs satisfy constraints (1) and (2).

We now prove that a solution of \( \text{MODEL}_{D,s,b} \) can only be an MFI or an MII depending on the value of \( b \). A solution of \( \text{MODEL}_{D,s,b} \) can only be frequent (i.e., \( b = \text{true} \)) or infrequent (i.e., \( b = \text{false} \)). If it is frequent then it is an MFI (see Definition 1). Otherwise it is an MII (see Definition 2). □
The propagator in Algorithm 1 enforces domain consistency on the constraint \(F \) remove 1 from \(i\). If so, we remove every item in \(\text{itemsets} satisfying the constraints we remove 1 from \(\text{dom}\) we first test if the itemset \(x\) cannot belong to a solution containing \(x\). Example 3.1 The lattice in Figure 1 displays the set of itemsets satisfying the constraints \(\text{FREQUENTSUBS}\) and \(\text{INFREQUENTSUPERS}\) on the dataset of Table 1 with \(s = 3\). The intersection between both sets is the set \(MFI \cup MILs\).

We can then use \(\text{MODELD} \cup b\) to mine either MFI or MILs by simply setting \(b\) to true or false. We now define the global constraints \(\text{FREQUENTSUBS}\) and \(\text{INFREQUENTSUPERS}\) and we propose propagators.

3.1 The Global Constraint FREQUENTSUBS

We present the new global constraint \(\text{FREQUENTSUBS}\) used to mine itemsets that have only frequent subsets.

Definition 5 (FREQUENTSUBS) Let \(x\) be a vector of Boolean variables, \(s\) a support threshold and \(D\) a dataset. The global constraint \(\text{FREQUENTSUBS}_{D,s}(x)\) holds if and only if \(\forall p \subseteq x^{-1}(1), \text{freq}(p) \geq s\).

Algorithm. The propagator for the global constraint \(\text{FREQUENTSUBS}\) is presented in Algorithm 1. Algorithm 1 takes as input the variables \(x\) and the support threshold \(s\). Algorithm 1 starts by computing the cover of the itemset \(x^{-1}(1)\) and stores it in \(\text{cover}\) (line 4). Then, for each item \(j \in x^{-1}(1), \) Algorithm 1 computes the cover of the subset \(x^{-1}(1) \setminus \{j\}\), and stores it in \(\text{cov}[j]\) (line 6). If \(x^{-1}(1) \setminus \{j\}\) is infrequent then \(x^{-1}(1)\) is not a solution and we return a failure (line 7). Algorithm 1 must then remove items \(i\) that cannot belong to a solution containing \(x^{-1}(1)\). To do that, we first test if the itemset \(x^{-1}(1)\) is infrequent (line 8). If so, we remove 1 from \(\text{dom}(x_i)\) for all \(i \in x^{-1}(s)\) (lines 9-10) because the itemset \(x^{-1}(1) \cup \{i\}\) has an infrequent subset \((x^{-1}(1))\) for every \(i \in x^{-1}(s)\). Otherwise, for every item \(i \in x^{-1}(s)\) we test if the itemset \(x^{-1}(1) \cup \{i\}\) is infrequent (line 12). If so, it could have infrequent subsets. Thus, for every item \(j \in x^{-1}(1), \) we test if the size of the cover of \(x^{-1}(1) \cup \{i\}\) \(\cap\) \(\{j\}\) (i.e., \(\text{cov}[j] \cap \text{cover}(i))\) is less than \(s\) (line 14). If so, we remove \(i\) from the possible items, that is, we remove 1 from \(\text{dom}(x_i)\) and break the loop (line 15).

Theorem 2 The propagator in Algorithm 1 enforces domain consistency on the constraint FREQUENTSUBS.

Algorithm 1: Propagator for FREQUENTSUBS

1. In: \(s\); support threshold;
2. InOut: \(x = \{x_1 \ldots x_n\}\); Boolean item variables;
3. begin
4. \(\text{cover} \leftarrow \text{cover}(x^{-1}(1));\)
5. \(\text{foreach} \ j \in x^{-1}(1) \text{ do}\)
6. \(\text{cov}[j] \leftarrow \text{cover}(x^{-1}(1) \setminus \{j\});\)
7. \(\text{if} \ |\text{cov}[j]| < s \text{ then return failure; }\)
8. \(\text{if} \ |\text{cover}| < s \text{ then}\)
9. \(\text{foreach} \ i \in x^{-1}(s) \text{ do}\)
10. \(\text{dom}(x_i) \leftarrow \text{dom}(x_i) \setminus \{1\};\)
11. \(\text{foreach} \ i \in x^{-1}(s) \text{ do}\)
12. \(\text{if} \ |\text{cover} \cap \text{cover}(i)| < s \text{ then}\)
13. \(\text{foreach} \ j \in x^{-1}(1) \text{ do}\)
14. \(\text{if} \ |\text{cov}[j] \cap \text{cover}(i)| < s \text{ then}\)
15. \(\text{dom}(x_i) \leftarrow \text{dom}(x_i) \setminus \{1\}; \text{ break;}\)

Proof. We first prove that if \(\text{FREQUENTSUBS}\) admits a solution, \(x^{-1}(1)\) is necessarily one of them. Suppose there is a solution and \(x^{-1}(1)\) is not one of them. This means that there exists a superset of \(x^{-1}(1)\) which has all its subsets frequent. \(x^{-1}(1)\) is one of these subsets. Thus, all subsets of \(x^{-1}(1)\) are frequent and \(x^{-1}(1)\) is solution too, which contradicts the assumption. We now prove that Algorithm 1 returns failure if and only if \(\text{FREQUENTSUBS}\) does not admit any solution. We know that \(\text{FREQUENTSUBS}\) has no solution if and only if \(x^{-1}(1)\) is not solution. \(x^{-1}(1)\) is not solution if and only if it has an infrequent subset \(x^{-1}(1) \setminus \{j\}\) for some \(j \in x^{-1}(1)\). In such a case the test in line 7 is true and failure is returned. If \(\text{FREQUENTSUBS}\) admits solutions, \(x^{-1}(1)\) is a support for \(x_i = 0\), for all \(i \in x^{-1}(s)\). As a result, Algorithm 1 does not need to check consistency of value 0 for any variable.

We now prove that Algorithm 1 prunes value 1 from \(\text{dom}(x_i)\) exactly when \(i\) cannot belong to a solution containing \(x^{-1}(1)\). Suppose value 1 of \(x_i\) is pruned by Algorithm 1. This means that the test in line 8 (or line 14) was true, that is, there exists a subset of \(x^{-1}(1) \cup \{i\}\) which is infrequent. Thus, by definition, \(x^{-1}(1) \cup \{i\}\) does not belong to any solution. Suppose now that value 1 of \(x_i\) is not pruned. From lines 8 and 14, we deduce that there does not exist any infrequent subset of \(x^{-1}(1) \cup \{i\}\). Thus, \(x^{-1}(1) \cup \{i\}\) is a solution and value 1 of \(x_i\) is domain consistent.

Theorem 3 Given a transaction dataset \(D\) of \(n\) items and \(m\) transactions, Algorithm 1 has an \(O(n^2 \times m)\) time complexity.

Proof. Computing the size of the cover of an itemset is in \(O(n \times m)\). Line 5 is called at most \(n\) times, leading to a time complexity in \(O(n^2 \times m)\). The time complexity of lines 8-10 is bounded above by \(n\). The test at line 13 is done at most \(n^2\) times. The cover \(x^{-1}(1) \cup \{i\} \setminus \{j\}\) at line 13 is computed in \(O(m)\) thanks to the cov data structure. Thus, the time complexity of lines 9-14 is bounded above by \(n^2 \times m\). As a result, Algorithm 1 has an \(O(n^2 \times m)\) time complexity.
Algorithm 2: Propagator for INFREQUENTSUPERS

1. In s: support threshold;
2. InOut: x = \{x_1 \ldots x_n\}: Boolean item variables;
3. begin
4. cover ← cover(x^{-1}(1) ∪ x^{-1}(s));
5. if |cover| ≥ s then
6.   foreach j ∈ x^{-1}(0) do
7.     if |cover ∩ cover(j)| ≥ s then
8.       return failure;
9. foreach i ∈ x^{-1}(s) do
10.    cover2 ← cover(x^{-1}(1) ∪ x^{-1}(s) \ \{i\});
11.   if |cover2| ≥ s then
12.     foreach j ∈ (x^{-1}(0) \cup \{i\}) do
13.       if |cover2 ∩ cover(j)| ≥ s then
14.         dom(xi) ← dom(xi) \ \{0\}; break;

3.2 The Global Constraint INFREQUENTSUPERS

We present the new global constraint INFREQUENTSUPERS used to mine itemsets that have only infrequent supersets.

Definition 6 (INFREQUENTSUPERS) Let x be a vector of Boolean variables, s a support threshold and D a dataset. The global constraint INFREQUENTSUPERS\_{D,s}(x) holds if and only if ∀p ⊆ x^{-1}(1), freq(p) < s.

Algorithm. The propagator for the global constraint INFREQUENTSUPERS is presented in Algorithm 2. Algorithm 2 takes as input the variables x and the support threshold s. Algorithm 2 starts by computing cover, the cover of the largest possible itemset, x^{-1}(1) ∪ x^{-1}(s) (line 4). If such an itemset has a frequent superset (line 7) then no itemset containing x^{-1}(1) can be a solution and we return failure (line 8). Algorithm 2 must then remove value 0 from dom(xi) if the absence of the item i can lead to an itemset that has a frequent superset. To do that, for every free item i (line 9), Algorithm 2 computes cover2, the cover of the largest itemset not containing the item i, i.e. x^{-1}(1) ∪ x^{-1}(s) \ \{i\} (line 10). If x^{-1}(1) ∪ x^{-1}(s) \ \{i\} is frequent (line 11), it could have a frequent superset. Thus, for every item j in x^{-1}(0) \cup \{i\} we test whether the itemset x^{-1}(1) ∪ x^{-1}(s) \ \{i\} \cup \{j\} is frequent (line 13). If so, we remove 0 from dom(xi) and break the loop (line 14).

Theorem 4 The propagator in Algorithm 2 enforces domain consistency on the constraint INFREQUENTSUPERS.

Proof. We first prove that if INFREQUENTSUPERS admits a solution, x^{-1}(1) ∪ x^{-1}(s) is necessarily one of them. Suppose there is a solution and x^{-1}(1) ∪ x^{-1}(s) is not one of them. This means that there exists a subset of x^{-1}(1) ∪ x^{-1}(s) which has all its supersets infrequent. x^{-1}(1) ∪ x^{-1}(s) is one of these supersets. Thus, all supersets of x^{-1}(1) ∪ x^{-1}(s) are infrequent and x^{-1}(1) ∪ x^{-1}(s) is solution too, which contradicts the assumption. We now prove that Algorithm 2 returns failure if and only if INFREQUENTSUPERS does not admit any solution. We know that if INFREQUENTSUPERS has solutions, x^{-1}(1) ∪ x^{-1}(s) is one of them. x^{-1}(1) ∪ x^{-1}(s) is not solution if and only if it has a frequent superset. In such a case the test in line 7 is true for some j in x^{-1}(0) and failure is returned. If INFREQUENTSUPERS has solutions, x^{-1}(1) ∪ x^{-1}(s) is a support for x_i = 1, for all i ∈ x^{-1}(s). As a result, Algorithm 2 does not need to check consistency of value 1 for any variable.

We now prove that Algorithm 2 prunes 0 from dom(x_i) exactly when i belongs to all solutions containing x^{-1}(1). If the test in line 13 is true this means that with x_i = 0 we get itemsets having at least a frequent superset. Hence, 0 must be pruned from dom(x_i). Suppose now that the test in line 13 is false for all j. x^{-1}(1) ∪ x^{-1}(s) \ \{i\} has only infrequent supersets and x_i = 0 is domain consistent.

Theorem 5 Given a transaction dataset D of n items and m transactions, Algorithm 2 has an O(n^3 \times m) time complexity. Proof. Computing the cover of x^{-1}(1) ∪ x^{-1}(s) in line 4 is in O(n \times m). The loop in line 6 computes the cover of x^{-1}(1) ∪ x^{-1}(s) \ \{j\} in O(m) using the already computed cover of x^{-1}(1) ∪ x^{-1}(s) (i.e., cover). As a result the worst case time complexity of lines 4-8 is in O(n \times m). Similary, the time complexity of lines 10-14 is bounded above by n \times m. Lines 10-14 are called at most n times, leading to a time complexity in O(n^2 \times m).

It is worth noticing the duality between Algorithm 1 and Algorithm 2. Algorithm 1 removes value 1 from dom(x_i) if including the item i leads to an itemset having an infrequent subset. Algorithm 2 removes value 0 from dom(x_i) if excluding the item i necessarily leads to itemsets having a frequent superset. This duality is not totally obvious at a first glance because Algorithm 1 requires the cov data structure to have its good time complexity whereas Algorithm 2 does not need it. The reason is that Algorithm 1 computes the covers of subsets that we cannot derive from the cover of their supersets whereas Algorithm 2 computes covers of supersets that can be obtained by simple bitwise operation on their subsets.

4 On Constrained Borders

As pointed out in [Bonchi and Lucchese, 2004], constraints can interfere with closedness (or maximality) when they are not anti-monotone. Likewise, constraints can interfere with min-
Given an itemset \( P \), a witness to its non maximality/closedness is an itemset \( P' \supseteq P \) that is frequent, satisfies \( C \), and in the case of closedness has the same frequency as \( P \). Checking that \( P' \) is frequent and checking that it has the same frequency as \( P \) is linear in \(|D|\). Checking that \( P' \) satisfies \( C \) requires the polynomial check of the \( C \) constraints. Hence, the "no" answer admits a polynomial certificate, and so deciding maximality or closedness is in \( \text{coNP} \).

**Corollary 1** Given a dataset \( D \) on a set of items \( I \) and a set \( C \) of user's constraints, finding an itemset that is maximal/closed/minimal/generator among those satisfying \( C \) is \( \text{coNP-hard} \).

## 5 Experiments

We made several experiments to compare our CP model \( \text{MODEL}_{D,s,b} \) to the state of the art approaches.

### 5.1 Experimental Protocol

The implementation of \( \text{MODEL}_{D,s,b} \) and its constraint propagators were carried out in the \texttt{Oscar} solver using Scala (bitbucket.org/oscarlib/oscar). The code is publicly available at gite.lirmm.fr/belaid/cp4borders. All experiments were conducted on an Intel core i7, 2.2Ghz with a RAM of 8Gb and a timeout of one hour. \( \text{MODEL}_{D,s,b} \) is denoted by \( \text{CP4MFI} \) (resp. \( \text{CP4MII} \)) when \( b = 1 \), i.e. mining MFIs, (resp. \( b = 0 \), mining MIIs). We used the global constraint \textsc{CoverSize} to encode the constraint (3) in \( \text{MODEL}_{D,s,b} \). \textsc{CoverSize}(\( x, p \)) holds if and only if \( p = \lvert \text{cover}(x^{-1}(1)) \rvert \) [Schaus et al., 2017]. We enforce frequency by simply adding the constraint \( p \geq s \) and infrequency by adding \( p < s \). The propagator of \textsc{CoverSize} enforces DC on the frequency, but it does not on the infrequency. We have assessed the quality of the \textsc{CoverSize} encoding of infrequency by comparing to a propagator that enforces DC on the infrequency. We observed that the number of additional nodes explored with the \textsc{CoverSize} encoding was marginal and the CPU time was better. We thus use the propagator of \textsc{CoverSize} in all our experiments. As an MFI is a closed itemset and an MII is a generator, we enhance propagation by adding, respectively, the global constraints \textsc{CoverClosure} [Schaus et al., 2017] and \textsc{Generator} [Belaid et al., 2019].

After a few preliminary tests, we decided to use smallest item frequency first as variable ordering heuristic and largest value first as value ordering heuristic. We compared \( \text{CP4MFI} \) to the \texttt{FPgrowth} [Grahne and Zhu, 2003] specialized algorithm for extracting MFIs (Borgelt’s platform: borgelt.net/fpgrowth.html) and \( \text{CP4MII} \) to \texttt{WALKY-G} [Stathammy et al., 2012] for mining MIIs (CORON platform: coron.loria.fr). We selected several real-sized datasets from the FIMI repository (fimi.ua.ac.be/data). A dataset is characterized by its name, the number of items \( |I| \), the number of transactions \( |T| \) and its density \( \rho \).

### 5.2 Mining Borders

Our first experiment compares \( \text{CP4MFI} \) to \texttt{FPgrowth} and \( \text{CP4MII} \) to \texttt{WALKY-G} for mining MFIs and MIIs. For each approach and each selected instance, Table 2 reports the CPU time and the number of MFIs/MIIs.\(^1\) An instance of a given dataset is characterized by its minimum support \( s \) (e.g., Zoo.50 denotes the instance of Zoo with \( s = 50 \)).

A first observation is that the specialized algorithms \texttt{FPgrowth} and \texttt{WALKY-G} follow a completely different approach to extract MFIs and MIIs whereas our CP model can

\(^1\)This number is computed by releasing the timeout.
Table 2: FPGrowth vs CP4MFI for mining MFI and WALKY-G vs CP4MFI for mining MIIs (time in seconds)

Table 3: FPGrowth vs CP4MFI for mining constrained MFI and WALKY-G vs CP4MFI for mining constrained MIIs (time in seconds)

Table 3: FPGrowth vs CP4MFI for mining constrained MFI and WALKY-G vs CP4MFI for mining constrained MIIs (time in seconds)

decreased $ub$ and we report the CPU time, in seconds, according to the number of solutions.

The main observation on the performance of FPGrowth for extracting constrained MIIs is that its CPU time is almost constant. For instance, extracting more than 5M or just 10 MFIIs on Chess_160 requires almost the same CPU time. This is explained by the fact that the cardinality constraint in FPGrowth has no pruning power. It is just used as a checker. On the contrary, when $lb$ increases, CP4MFI removes more values thanks to constraint propagation, thus drastically reducing the search space. Take for instance Pumsb_17000. When $lb = 22$, CP4MFI extracts the 55K MIIs in more than 50 minutes whereas when $lb$ increases to 30, CP4MFI returns the 21 remaining solutions in only 15 seconds.

On instances with no solutions, such as Pumsb_17000 with $lb = 35$, CP4MFI proves the absence of solutions in one second whereas FPGrowth spends 157.28 seconds.

For WALKY-G, no results are reported because it already has an OOM or T0 state before reaching post-processing step. The main observation that we can draw on the behavior of CP4MFI is again the strong correlation between the increase of tightness of the cardinality constraint (that is, the decrease of $ub$) and the drop in CPU time needed to extract the constrained MIIs. The explanation for this good behavior of CP4MFI is again the strength of constraint propagation to discard inconsistent values, thus reducing the search space. On Pumsb_17000, when $ub$ decreases from 8 down to 1, CP4MFI goes from 45 minutes down to 4.59 seconds to return the set of MIIs (1.3M MIIs for $ub = 8$ and 2K MIIs for $ub = 1$).

6 Conclusion

We have presented a CP model for mining MIIs and MIIs. We defined two new global constraints, namely FREQSUBS and INFREQSUPERs, with polynomial complete propagators. We have proved that the problem of MIIs or MIIs with constraints is co-NP-hard, ruling out the hope for a single CSP solving this problem. Nevertheless, our CP model is sound when the additional constraints are monotone on MIIs or anti-monotone on MIIs. Experiments showed that the CP approach is competitive with state of the art techniques for mining MIIs or MIIs and even better for mining MIIs or MIIs with additional constraints.
References


