# CoverSize: A Global Constraint for Frequency-based Itemset Mining

Pierre Schaus[1] and John O.R. Aoga[1,2](0000-0002-7213-146X) and Tias Guns[3]

[1]UCLouvain, ICTEAM (Belgium); [2]UAC, ED-SDI (Benin)
[3]VUB Brussels (Belgium) and KU Leuven (Belgium)
{john.aoga,pierre.schaus}@uclouvain.be; tias.guns@{vub.be,cs.kuleuven.be}

**Abstract.** Constraint Programming is becoming competitive for solving certain data-mining problems largely due to the development of global constraints. We introduce the CoverSize constraint for itemset mining problems, a global constraint for counting and constraining the number of transactions covered by the itemset decision variables. We show the relation of this constraint to the well-known table constraint, and our filtering algorithm internally uses the reversible sparse bitset data structure recently proposed for filtering table. Furthermore, we expose the size of the cover as a variable, which opens up new modelling perspectives compared to an existing global constraint for (closed) frequent itemset mining. For example, one can constrain minimum frequency or compare the frequency of an itemset in different datasets as is done in discriminative itemset mining. We demonstrate experimentally on the frequent, closed and discriminative itemset mining problems that the CoverSize constraint with reversible sparse bitsets allows to outperform other CP approaches.

## 1 Introduction

Frequent itemset mining (FIM) is one of the well-known and most studied data mining problems [8] and first introduced in [2]. Guns et al. [19] showed that FIM problems could be modelled and solved using Constraint Programming (CP) with the additional benefit that new constraints can easily be integrated into the models. Since then several CP (also SAT) approaches have been proposed for other data-mining problems such as frequent sequence mining [22,29], dominance-based pattern mining [28] and closed FIM [20,21,24].

The flexibility of adding constraints when using a generic CP solver typically comes at the cost of efficiency; a well-known tradeoff. We can hence look at itemset mining papers in terms of where they are on the efficiency versus generality scale. Most works in itemset mining focus primarily on efficiency [8,39], while typical constraint-based mining papers hard-code a select number of constraints based on properties like (anti-)monotonicity [32]. Earlier papers on using CP for itemset mining focus mostly on generality and decompose the itemset mining constraints into many (reified) linear constraints [19] at the cost of efficiency. In line with recent works in CP for sequence mining [3,4,22], Lazaar et al. [24] have

shown that a single global constraint for closed frequent itemset mining can outperform a decomposition approach. This comes at significant cost for generality though, because 1) by encapsulating all but the itemset variables, only syntactic constraints on the items can be added; 2) only *closed frequent* patterns can be found and adding syntactic constraints can have unwanted side-effects [6].

In this paper, we aim to maximize both generality and efficiency while employing a global constraint for itemset mining. We achieve this by introducing the *CoverSize* global constraint, which 1) computes a lower and upper-bound on the frequency and synchronizes this with a decision variable, meaning that the frequency can be used in other separate constraints; 2) internally, the filtering algorithm uses the reversible sparse bitset data structure which was introduced to efficiently filter table constraints [14]. This *CoverSize* constraint can be combined with a *CoverClosure* constraint to enforce the closed property [24,34] or with a discriminative optimization constraint (such as $\chi^2$) to solve the correlated itemset mining problem as in [31].

In contrast to most constraints in CP, what is typical about global constraints for data mining is that they must be able to handle large amounts of data. A traditional global constraint that shares this property is the table constraint, which has a rich history in CP literature [5,12,25,35]. Its link with a global constraint for itemset mining (IM) is even stronger, as both can be seen as operating on a binary matrix; for IM the columns are *items* (Boolean variables) and for table the columns are *(variable, value) pairs*. The use of bitvectors and fast bitvector operations is common in itemset mining implementations, indeed, it was also used for the closed FIM constraint [24]. Related, a column-based bitvector representation for the table constraint was recently proposed [14], and the propagator was shown to outperform all other approaches. Inspired by this relation, we show that the *reversible sparse bitset* data structure that was devised for table can also be used to implement efficient itemset mining propagators. Our scaling experiments on large and sparse data indicate the benefit of this.

Furthermore, our proposed *CoverSize* global constraint propagates from the *item* variables to a variable representing the frequency and back. This means that the same constraint can be used to enforce a minimum and maximum frequency of a set in a table/database. We show that domain-consistent filtering is NP-hard, though good results can be obtained with weaker filtering. We showcase the added flexibility of such a choice by using it as a building block for modelling closed frequent itemset problem [24] and correlated itemset mining problem [31]. For *closed*, we argue for and propose a separate global constraint.

Our contributions are hence as follows: we show how advances in data structures for table constraints can benefit global constraints for itemset mining too; we propose that global constraints for itemset mining expose the frequency through a variable, and demonstrate how this allows, for example, to solve discriminative itemset mining too; and empirically our experiments with a generic CP solver show that this approach outperforms other CP approaches and is on par with a special-purpose CP solver, thereby decreasing the gap to the highly efficient specialized itemset miners.

## 2 Background

### 2.1 Itemset Mining

Frequent itemset mining is concerned with finding a set of *items* that appears frequently in a database of sets [2]. The database is often called a transaction database, and each entry in the database is called a transaction (such as a purchase of products). Itemset mining has applications in market basket analysis, web log mining, bio-informatics and more [1].

More formally, given a set $\mathcal{I}$ of $n$ possible items and a transaction database of size $m$: $\mathcal{H} = \{(t, T) \mid t \in \{1, \ldots, m\}, T \subseteq \mathcal{I}\}$. Figure 1a shows an example database with $\mathcal{I} = \{A, B, C, D\}$. The goal of the *frequent itemset mining problem* is to enumerate all sets $I \subseteq \mathcal{I}$ such that $|\{(t, T) \in \mathcal{H} \mid I \subseteq T\}| \geq \theta$ with $\theta$ a user-supplied threshold.

The set of transactions that contain the itemset $\{(t, T) \in \mathcal{H} \mid I \subseteq T\}$ is called the *cover*. Computing this set efficiently is a core aspect of itemset mining algorithms. Different algorithms have used different representations of the transaction database. Figure 1(a,b,c) shows three of them. In a vertical representation, the intersection is the key operation. Let $\mathcal{V}(i)$ be the set of transaction identifiers of item $i$, then $|\{(t, T) \in \mathcal{H} \mid I \subseteq T\}| = |\{\bigcap_{i \in I} \mathcal{V}(i)\}|$. In case of a vertical dense bitvector representation, efficient bitwise operations can be used for the intersection, which scales very well in practice.

Many other variations on frequent itemset mining have been investigated. For example, *closed* frequent itemset mining adds the additional restriction that an itemset must not have a superset with the same frequency: $\nexists I' \supset I : \{(t, T) \in \mathcal{H} \mid I' \subseteq T\} = \{(t, T) \in \mathcal{H} \mid I \subseteq T\}$; and *maximal* frequent itemset mining has additional restriction that an itemset must not have any frequent superset: $\nexists I' \supset I : |\{(t, T) \in \mathcal{H} \mid I' \subseteq T\}| \geq \theta$ [1]. Other constraints on items and transactions have been investigated as well [32].

In an optimization setting, one can search for the most or least frequent itemsets (typically under a number of other constraints) or find the most discriminating itemsets. Given two databases $\mathcal{H}^+$ and $\mathcal{H}^-$ (for example, from two consecutive months), the goal is to find the itemset(s) that best discriminate between the two; such as an itemset that is very frequent in one and barely frequent in the other. A range of *discriminative* measures, also called correlation measures, have been studied [27]. A property that we will exploit later is that these measures can be computed using just information on the frequency of the sets plus the total number of transactions of the databases. We can hence denote these measures by a function $f(|\mathcal{H}^+|, |\mathcal{H}^-|, p, n)$ where $p, n$ represents the frequency of the itemset in the two databases.

*Modeling itemsets.* Following [13], we use an array of Boolean decision variables $\mathcal{I} = [I_1, I_2, \ldots, I_n]$ to represent an itemset $X \subset \mathcal{I}$. Each $I_i$ is a binary variable with domain $dom(I_i) = \{0, 1\}$ and an item $i \in X \iff I_i = 1$. We say that $I_i$ is *unbound* if there is more than one value in $dom(I_i)$. $I_i$ is *bound* to 1 (0) means the item $i$ is part (not part) of the itemset. Hence, one assignment to $\mathcal{I}$ corresponds to one itemset.

The *decomposition* formulation of frequent itemset mining [19] introduces an extra array of Boolean decision variables $\mathcal{T} = [T_1, T_2, \ldots, T_m]$, one for each of the $m$ transactions. A Boolean variable $T_t$ indicates whether the transaction with identifier $t$ belongs to the cover $\{(t, S) \in \mathcal{H} \mid I \subseteq S\}$. This is enforced with a constraint for every transaction as follows: $\forall (t, S) \in \mathcal{H} : T_t = 0 \iff \bigvee_{i \notin S} I_i$. In other words: if an item $i$ is in the itemset and not in the transaction $(t, S)$ then this transaction is not covered by the itemset and equivalently if a transaction is not covered none of the items $i$ in the itemset do belong to $(t, S)$. The size of the cover can then be constrained as follows: $\sum_t T_t \geq \theta$. This model is not domain consistent for the frequent itemset mining problem that aims to enumerate all frequent patterns for a certain $\theta$. As suggested in [13], one can further add the redundant constraints $\forall i : I_i = 1 \implies (\sum_{(t,S)\in\mathcal{H}, i\in S} T_t) \geq \theta$ to achieve domain consistency for the frequent itemset problem: these constraints enforce that an item is only supported if adding it to the current itemset will not violate the frequency constraint.

### 2.2 Table Constraint and Reversible Sparse Bit-Sets

A table constraint enforces that an array of integer decision variables $[V_1, \ldots, V_n]$ corresponds to one of the provided tuples $\Gamma = \{(t, \tau)\} | t \in \{1, \ldots, m\}\}$, where $t$ is the tuple identifier and each tuple $\tau = (v_1, \ldots, v_n)$ consists of $n$ values corresponding to the $n$ variables: $table([V_1, \ldots, V_n], \Gamma) \iff \exists (t, \tau) \in \Gamma : V_1 = \tau_1 \wedge \ldots \wedge V_n = \tau_n$. A key property to maintain is the set of tuples supported by the current domain: $currTable = \{(j, \tau) \in \Gamma \mid \tau_1 \in dom(V_1) \wedge \ldots \wedge \tau_n \in dom(V_n)\}$. In [14], a reversible sparse bitset was proposed to maintain the set of tuple indices during search. In the propagator, a dense vertical representation of $\Gamma$ is used: for every variable/value combination $(V_i, v), v \in dom(V_i)$, a bitvector $support[V_i, v] = \{(j, \tau) \in \Gamma \mid \tau_i = v\}$ is precomputed that stores the tuple identifiers in which the pair $(V_i, v)$ appears. The indices of $currTable$ and the consistency of each $(V_i, v)$ is computed using bitwise operations, e.g. $(V_i, v)$ is supported if $support[V_i, v] \cap currTable \neq \emptyset$.

We briefly recall the `RSparseBitSet` data structure [14] which we will use in our propagators. The pseudo-code of this data structure is given in Algorithm 1 and some illustrative methods are also shown. The Reversible Sparse BitSet represents a set as a bitset (array of 64-bit Long *words*) and is *"reversible"* means that it is able to restore itself on backtrack. The reversibility relies on a global trail mechanism well known in the folklore of constraint programming (see [23] for an introduction to trailing and time-stamping).

The originality of this structure is that it borrows the idea of reversible sparse-sets [37] to discard all-zero words. When a bitvector is sparse (contains many zero words), this can save unnecessary iterations and computations over those words.

The following class invariant is maintained to ignore zero words: the number of non-zero words is a reversible integer denoted `limit`; and the `limit` first entries of `index` are indexes to the non-zero words in the bitvector. All the words beyond that limit are the indexes of zero words.

For the intersect method, which is also crucial for itemset mining, one can see how this is maintained by exchanging a detected zero word with the last non-zero one before decreasing the `limit` (swapping).

Apart from skipping entire words, the bitvector representation allows using highly efficient operations over entire words such as *and* and *bitCount*.

## 3 Global constraints for frequency-based itemset mining

There is a close relation between a table constraint that reasons over a binary representation of the table and itemset mining. Each variable/value pair $(V_i, v)$ is a column and can be seen as an *item* (in the itemset mining problem), and internally a vertical dense representation of the table can be used. Because each tuple in table $\Gamma$ is of size $n$, in a binary representation of the table there will be exactly $n$ non-zero entries per row. Further knowing that there are exactly $n$ variables that each must be assigned one value, one can see that checking whether the set representation of $V : \{(V_i, v) \mid V_i = v \in V\}$ is a *subset* of the set representation of a tuple $\tau : \{(V_i, \tau_i) \mid \tau_i \in \tau\}$ coincides with checking whether they can be *equal* as both sets will have equal length when $V$ is fully assigned. The cover relation of itemset mining is hence equivalent to the table support relation in this case, and the table constraint can be seen as enforcing a minimum frequency constraint with $\theta = 1$.

Earlier work has proposed a single global constraint for minimum frequent closed itemset mining. For efficiency reasons, we propose to use the reversible sparse bitset to maintain the set of transactions that can still be covered. For generality reasons, we propose to separate the computation of the frequency from the minimum (or maximum) frequency restriction and to separate that from enforcing the closedness property.

### 3.1 Computing frequency: the *CoverSize* constraint

Given a set of boolean variables $\mathcal{I}$ representing the pattern (selected items), a vertical dense bitvector representation of the database $\mathcal{D}$ (see Fig. 1c for an example), and an integer variable $c$, the *CoverSize* global constraint enforces the relation

$$\mathit{CoverSize}([I_1, \ldots, I_n], \mathcal{D}, c) \iff c = \left| \bigcap_{I_i=1} \mathcal{D}(I_i) \right|$$

such that $c$ represents the number of bits set in the intersection of the vertical bitvectors ($\mathcal{D}$) of the selected items. Using bitwise operations it can be formulated as

$$\mathit{CoverSize}([I_1, \ldots, I_n], \mathcal{D}, c) \iff c = \texttt{size}(\&_{I_i=1}\mathcal{D}(I_i)).$$

For example in Fig. 1c and for itemset $\{C, D\}$, $c = |\mathcal{D}(I_C)\ \&\ \mathcal{D}(I_D))| = 2$.

Lazaar et al.[24] have argued that a global constraint is preferred over a decomposition into a constraint per transaction because the many constraints that need to be handled create overhead for the solver. This was shown earlier

in [30], which proposed a CP-inspired dedicated solver with a global constraint for (reified) matrix-wide operations over bitvector variables.

When not exporting the cover as individual Boolean variables, we can use an internal data structure to store the cover such as `RSparseBitSet`. Note that not exposing the cover also limits the generality of the approach: no constraints can be put on the cover so that constraints such as closedness, maximality, non-frequency-based quality measures, etc either require changes to the global constraint, or a separate global constraint that recomputes the cover. However, there are a number of constraints that depend only on the size of the cover and hence for added flexibility we propose to hide the cover but expose the cover size.

**Consistency of *CoverSize*** Theorem 1 is used to demonstrate that it is NP-hard to check the consistency for *CoverSize*.

**Theorem 1.** *Given a collection of sets $\{S_1, \ldots, S_n\}$, the problem of finding a subset of these such that their union is of fixed cardinality $k$ is NP-hard.*

*Proof.* We build a reduction from the NP-hard exact cover by three sets (X3C) problem: given a collection $\{C_1, C_2, \ldots, C_n\}$ of 3-element subsets built from a universe $X$ with $|X| = 3q$ (a multiple of 3), can we find exactly $q$ subsets of $C$ to cover $X$? We reduce this X3C problem into our problem such that $S_i = C_i \cup \{a_1^i, \ldots, a_{|X|+1}^i\}\ \forall i \in \{1, \ldots, n\}$. All the artificial $a_j^i$ elements added are different and not in universe $X$. Each set $S_i$ has thus a cardinality of $3+|X|+1 = 4+3q$. We are looking for a collection of sets such that their union is of size $k = q(4+3q)$. Only $q$ sets can be selected; even when counting just the artificial elements ($|X|+1 = 3q+1$ per set), more than $q$ sets is not possible because $k - ((q+1)\cdot(3q+1)) < 0$. Fewer then $q$ sets is also not possible because $k - ((q-1)\cdot(3q+1)) > |X|$ and hence there would need to be more than $|X|$ unique elements in universe $X$ to achieve cardinality $k$. For exactly $q$ sets, one can verify that these $q$ sets will cover $|X|$ after the removal of the $q(|X|+1)$ added elements: $q(4+3q) - q(|X|+1) = q(4+3q-3q-1) = 3q = |X|$. ■

**Corollary 1.** *Given a collection of sets $\{S_1, \ldots, S_n\}$, the problem of finding a subset of these such that their intersection is of fixed cardinality $k$ is NP-hard.*

*Proof.* We reduce the problem of Theorem 1. Let $X = \bigcup_i S_i$ be the universe and $\overline{S}_i = X \setminus S_i$ the complement set of $S_i$ w.r.t. $X$. There exists a subset $\Omega \subseteq \{1, \ldots, n\}$ such that $\left|\bigcup_{i\in\Omega} S_i\right| = k$ if and only if $\left|\bigcap_{i\in\Omega} \overline{S}_i\right| = |X| - k$. ■

**Theorem 2.** *Determining the satisfiability for CoverSize is NP-hard.*

*Proof.* The problem of Corollary 1 is reduced to finding a feasible solution for $\mathit{CoverSize}([I_1, \ldots, I_n], \mathcal{D}, c = k)$ with $\mathcal{D}(I_i)$ the bitvector representation for set $S_i$. ■

Despite this hardness result, we can still propagate many conditions efficiently. The hardest part is to propagate from an upper bound on $c$ to the item variables.

***CoverSize* Propagator** We denote by $U = \{I_i \in \mathcal{I} \mid dom(I_i) = \{0,1\}\}$ the set of undecided items and by $P = \{I_i \in \mathcal{I} \mid dom(I_i) = \{1\}\}$ the set of included items. The filtering rules for *CoverSize* are:

1. (*Rule* 1) computes the maximum cover size (exact upper-bound) that corresponds to discarding all the undecided items: $\max(c) \leq \left|\bigcap_{I_j \in P} \mathcal{D}(I_j)\right|$.
2. (*Rule* 2) computes the minimum cover size (exact lower-bound) that corresponds to including all the undecided items $\min(c) \geq \left|\bigcap_{I_j \in (P \cup U)} \mathcal{D}(I_j)\right|$.
3. (*Rule* 3) discards item $I_i$ if including it would result in a cover size that is below the minimum threshold. $\forall I_i \in U : \left|\bigcap_{I_j \in P} \mathcal{D}(I_j) \cap \mathcal{D}(I_i)\right| < \min(c) \implies I_i = 0$. This rule is also implemented in [24] and can be achieved in the decomposition with a redundant constraint for each item separately.
4. (*Rule* 4) detects mandatory items. If the lower-bound is equal to the maximum allowed cover size, then if the cover size lower-bound would increase while excluding an item $I_i$ then this item $I_i$ is mandatory. $\forall I_i \in U : \left|\bigcap_{I_j \in (P \cup U)} \mathcal{D}(I_j)\right| = \max(c) \wedge \left|\bigcap_{I_j \in (P \cup U)} \mathcal{D}(I_j)\right| < \left|\bigcap_{I_j \in (P \cup U) \setminus \{I_i\}} \mathcal{D}(I_j)\right| \implies I_i = 1$.

Algorithm 2 gives the filtering algorithm for the *CoverSize* constraint implementing the Rules 1-4. $N$ denotes the newly bound item variables since the previous call to the `propagate` method. The algorithm is thus incremental.

The block at Line 5 updates the current cover to reflect the new items included in the itemset[1]. The second block at Line 8 filters out the items that if included would induce a cover size below the allowed threshold $\min(\texttt{c})$. This corresponds to Rule 3.

Line 11 computes the upper-bound of the cover size according to Rule (1). The lower-bound (Line 12) is obtained by including all the unbound items according to Rule (2).

Line 13 is triggered when the smallest *possible* intersection size (*lb*) is the largest *allowed* size of the frequency variable ($\max(\texttt{c})$). In this case, all the items that are mandatory to reach the lower-bound can be forcefully included (this is not necessarily true when $\min(\texttt{c}) = \max(\texttt{c})$ as $\min(\texttt{c})$ can be externally set resulting in $\min(\texttt{c}) > \texttt{lb}$). An unbound item $I$ is mandatory if it is the only item that does *not* contain a transaction that all the other unbound and included ones do; in that case `lb` would increase to $\texttt{lb}' > \max(\texttt{c})$. In the algorithm, $\texttt{m} \leftarrow \texttt{cover}\ \&_{I_j \in U \setminus I_i} \mathcal{D}(I_j)$ is the cover if one would include all unbound items except $I$. If $\texttt{m} \nsubseteq \mathcal{D}(I_i)$ then $\texttt{m} \cap \mathcal{D}(I_i)$ would be a smaller set than $m$ and hence $I_i$ is mandatory to obtain the smallest cover size *lb*. For the example of Fig. 1, let $C$ be included then $\texttt{lb} = 1, \texttt{ub} = 3$. Let $\max(\texttt{c}) = 1$ then $A$ is mandatory: not including it results in $m = \{2,4\}, m \nsubseteq \{1,3,4\}$ because of transaction 2. This condition is equivalent to Rule 4 but slightly more efficient to compute as it does not require to consider every non-zero words by returning true as soon as one word of `m` is not included.

[1] This is similar to the update of *currTable* in [14] for filtering table constraints.

**Algorithm 1:** Class `RSparseBitSet`. t[0] denotes the first element of array t and $0^k$ denotes a sequence of $k$ bits set to 0.

```
1  words: array of rlong          // reversible longs, array length = p
2  index: array of int                                  // array length = p
3  limit: rint                                  // a reversible integer
4  Method intersect(m: array of long)
       /* this ← this & m                                              */
5      foreach i from limit downto 0 do
6          o ← index[i]
7          w ← words[o] & m[o]                              // bitwise AND
8          words[o] ← w
9          if w = 0^64 then
10             swap(index[i], index[limit])
11             limit ← limit − 1
12 Method contains(m: array of long): bool
       /* m ⊆ this                                                     */
13     foreach i from 0 to limit do
14         o ← index[i]
15         if (words(o) & ~ m[o]) ≠ 0^64 then
16             return false
17     return true
```

**Algorithm 2:** Class CoverSize($[I_1, \ldots I_n], \mathcal{D}, c$)

```
1  cover: RSparseBitSet                                  // Current cover
2  N, U                    // New bound variables, Unbound variables
3  D                                        // D[I_i] = bit-set for item I_i
4  Method propagate()
       /* update current cover                                         */
5      foreach variable I_i ∈ N do
6          if I_i = 1 then
7              cover ← cover & D[I_i]
       /* remove items that if included induce cover < min(c)          */
8      foreach variable I_i ∈ U do
9          if size(cover & D[I_i]) < min(c) then
10             I_i ← 0
       /* cover bounds                                                 */
11     ub ← size(cover); max(c) ← min(max(c), ub)
12     lb ← size(cover &_{I_i ∈ U} D[I_i]); min(c) ← max(min(c), lb)
       /* propagating maximum size                                     */
13     if lb < ub ∧ lb = max(c) then
14         foreach variable I_i ∈ U do
               /* include items mandatory for a cover size = lb        */
15             m ← cover &_{I_j ∈ U \ I_i} D[I_j]
16             if m ⊈ D[I_i] then
17                 I_i ← 1
```

The time complexity for executing propagate is $O(|\mathcal{I}| \times m/64)$ with $|\mathcal{I}|$ the number of items and $m/64$ the number of words necessary to represent the cover. In practice, the reversible sparse bitset will only iterate on the non-zero words in the cover bitvector. The space complexity is $O(|\mathcal{I}| \times m)$ similar to that of other approaches and due to the space needed to store the database.

Since domain consistency *CoverSize* is NP-hard we can unfortunately not clearly characterize[2] the filtering of Algorithm 2. Only in the case of an unconstrained max(c) (for instance for the frequent itemset problem), the filtering reaches domain consistency.

### 3.2 Closed itemsets: the *CoverClosure* constraint

The idea of mining for *closed* frequent itemsets is to reduce the set of extracted itemsets to a smaller, more interesting one. The intuitive idea is that if a frequent pattern has a cover that is exactly the same as a super pattern, then only the super pattern should be enumerated.

An itemset is hence a *closed* itemset if there is no superset with the same cover: $\nexists I' \supset I : \{(t,T) \in \mathcal{H} \mid I' \subseteq T\} = \{(t,T) \in \mathcal{H} \mid I \subseteq T\}$. Hence, the *closure* of an itemset can be computed by verifying which items could be added to the itemset without changing the cover:

$$clo_{\mathcal{H}}(I) = I \cup \{j \notin I \mid \{(t,T) \in \mathcal{H} \mid I \cup \{j\} \subseteq T\} = \{(t,T) \in \mathcal{H} \mid I \subseteq T\}\} \quad (1)$$

As argued in [6] there are two ways of interpreting the *closed* property when combined with other constraints: 1) of all closed itemsets, keep only those that satisfy the constraints 2) take the closure such that the new itemset satisfies all constraints

This can have far-reaching consequences. Let us take the database in Fig. 1, where item $B$ is in all transactions and so will be in all closed itemsets. If we now add the constraint $'B' \notin I$, then under interpretation 1 there would not be any valid itemset, while under interpretation 2 there is $D, C, AD$ and $ACD$ namely all closed itemsets when ignoring $B$. It should be clear that interpretation 1) should not be taken by default. On the other hand, enforcing interpretation 2) requires one to reason not in terms of local constraints but over valid solutions to the CSP, for example with dominance properties [28]. Nonetheless, interpretation 1) is valid as long as all constraints have the property that adding a cover-preserving item to the set can never violate another constraint; for example, one can freely add a minimum size or maximum frequency constraint [6]. In case of such constraints it must be expressed as a preference over solutions of the CSP, e.g. by adding constraints each time a solution is found [28]. We hence propose to offer the widely used unconstrained closure operator as a separate *CoverClosure* constraint.

Another argument for separating the closure constraint is that in case of discriminative itemset mining, we may want to enforce closedness on only one of the two databases or on the entire database [19]. A separate constraint allows this freedom.

[2] As for many NP-hard global constraints like bin-packing, cumulative, circuit, etc.

***CoverClosure* Propagator** Two filtering rules are enforced similar to [24]:

1. (*Rule* 5) Closure inclusion. This rule checks for each unbound item $I_i$ if including it would result in an unchanged cover. If yes, this item should be included in the final pattern. More formally

$$\forall I_i \in U : (\bigcap_{I_j \in P} \mathcal{D}(I_j) \cap \mathcal{D}(I_i)) = \bigcap_{I_j \in P} \mathcal{D}(I_j) \Longrightarrow I_i = 1$$

2. (*Rule* 6) Closure exclusion. This rule detects if extending the pattern with an item would result in a cover for which there is an already excluded item that should be added by the closure operator. Hence, including the first item would lead to an inconsistency and so it should be excluded. More formally, assuming $I_k \in \mathcal{I}, I_k = 0$ represents the excluded items:

$$\forall I_i \in U, I_k \in \mathcal{I}, I_k = 0 : ((\bigcap_{I_j \in P} \mathcal{D}(I_j)) \cap \mathcal{D}(I_i)) \subseteq ((\bigcap_{I_j \in P} \mathcal{D}(I_j)) \cap \mathcal{D}(I_k)) \Longrightarrow I_i = 0$$

Algorithm 3 implements the domain consistent filtering for the *CoverClosure* constraint. This constraint also uses the `RSparseBitSet` data structure to store the cover. It has a complexity of $O(|\mathcal{I}|^2 \times \ m/64)$.

A faster (but not domain consistent) filtering is obtained by replacing Rule 6 with a consistency check verifying that for each discarded item ($I_i = 0$), including it changes the cover: $\forall I_k \in \mathcal{I}, I_k = 0 : (\bigcap_{I_j \in P} \mathcal{D}(I_j) \cap \mathcal{D}(I_k)) = \bigcap_{I_j \in P} \mathcal{D}(I_j) \Longrightarrow fail$. This version has a complexity of $O(|\mathcal{I}| \times \ m/64)$, similar to *CoverSize*.

## 4 Frequency-based itemset mining with *CoverSize* and *CoverClosure*

### 4.1 Frequent itemset mining

Our model for frequent itemset mining contains just one *CoverSize* and a constraint that the size of the cover is above a fixed minimum frequency $\theta$:

$$\text{enumerate} \quad \mathit{CoverSize}([I_1, \ldots, I_n], \mathcal{D}, c) \wedge c \geq \theta$$

Notice that as $c$ is a variable, one can also add a maximum frequency constraint, or use it in branch-and-bound to search for the most frequent itemsets under constraints such as a minimum itemset cardinality:

$$\text{maximize} \quad c, \quad \text{s.t.} \quad \mathit{CoverSize}([I_1, \ldots, I_n], \mathcal{D}, c) \wedge \sum_i I_i \geq \beta$$

### 4.2 *Closed* frequent itemset mining

Looking for the frequent closed itemset amounts to adding *CoverClosure*:

$$\text{enumerate} \quad \mathit{CoverSize}([I_1, \ldots, I_n], \mathcal{D}, c) \wedge c \geq \theta \wedge \mathit{CoverClosure}([I_1, \ldots, I_n], \mathcal{D})$$

As explained in Sect. 3.2, other constraints should only be added if they do not constrain the addition of (frequency-preserving) items.

**Algorithm 3:** Class CoverClosure($[I_1, \dots I_n], \mathcal{D}$)

```
1  cover: RSparseBitSet                                  // Current cover
2  N, U                    // New bound variables, Unbound variables
3  D                                          // D[I_i] = bit-set for item I_i
4  Method propagate()
       /* update current cover                                          */
5      foreach variable I_i ∈ N do
6          if I_i = 1 then
7              cover ← cover & D[I_i]
       /* Rule 5                                                        */
8      foreach variable I_i ∈ U do
9          if cover = (cover & D[I_i]) then
10             I_i ← 1
       /* Rule 6                                                        */
11     foreach variable I_i ∈ U do
12         foreach variable I_k ∈ I with I_k = 0 do
13             if (cover & D[I_i]) ⊆ (cover & D[I_k]) then
14                 I_i ← 0; break
```

### 4.3 Discriminative (closed) itemset mining

Given a split database $\mathcal{D}^+$ and $\mathcal{D}^-$ containing positive (+) and negative (−) transactions defined on a same set of items, the objective is to find the highest scoring itemsets (discriminating one class over another) w.r.t. a correlation (discriminative) measure such as accuracy, information gain, $\chi^2$ measure, Gini index, etc. Those itemsets are interesting as classification rules directly[9,11,16], or as features (if the itemset is present or not) for another classifier[10,15].

Using *accuracy* as a discriminative measure leads to the following problem:

$$\text{maximize} \quad p - n, \quad \text{s.t.}$$
$$\mathit{CoverSize}([I_1, \dots, I_n], \mathcal{D}^+, p) \wedge \mathit{CoverSize}([I_1, \dots, I_n], \mathcal{D}^-, n)$$

Another standard discriminative measure is the $\chi^2$ one depicted on Fig.1d. As explained in [31] the standard discriminative functions such as $\chi^2$ have the property that they are zero on the diagonal (relative to the possible values of $p, n$) and convex (denoted by ZDC). A general ZDC-based model for discriminative itemset mining is composed of

- two constraints $\mathit{CoverSize}([I_1, \dots, I_n], \mathcal{D}^+, p)$ and $\mathit{CoverSize}([I_1, \dots, I_n], \mathcal{D}^-, n)$ to compute the cover size on the positive and negative transactions;
- a Zero Diagonal Convex constraint $\mathit{ZDC}(|\mathcal{D}^+|, |\mathcal{D}^-|, p, n, \mathit{score})$ that links $p$, $n$ and *score* using a discriminative function (such as $\chi^2$) to maximize;
- optionally $\mathit{CoverClosure}([I_1, \dots, I_n], \mathcal{D}^- \cup \mathcal{D}^+)$ to obtain the closedness property. Note that posting *CoverClosure* separately on the positive (negative) can decrease $p$ ($n$) and is hence not allowed for symmetric ZDC measures.

This approach which employs a separate *ZDC* constraint that takes only the cardinalities $p$ and $n$ as input, is novel and favors reusability in a different context: it is itemset-agnostic, meaning that it could also be used for example to find discriminating sequences instead of itemsets. In [31] the authors also employ a global constraint for the discriminative itemset mining problem, but one that reasons at the transaction level with one variable per transaction. The filtering they achieve is stronger than our decomposition into three constraints. They perform what they call a redundant *look-ahead* filtering[3] on each item separately.

We briefly describe our filtering for *ZDC*, illustrated in Fig.1e. Because of the ZDC property, the minimum and maximum is located at one of the four corners of the box $[\min(p), \max(p)] \times [\min(n), \max(n)]$, and hence only these extremes need to be computed for pruning $\min(score)$. Given a minimum value for $\min(score)$, for example as enforced during branch-and-bound maximization, the value of $\max(p)$ and $\max(n)$ can be reduced as illustrated on Fig.1e. On this figure, the iso-curve corresponding to $\min(score)$ is visualized. The ZDC property implies that any larger score must lay outside of the region enclosed by the iso-curves. The gray zone on Fig.1e corresponds to inconsistent combinations for $p$ and $n$, hence discovering the new minimum for $p$ requires to find $v$ such that $ZDC(|\mathcal{D}^+|, |\mathcal{D}^-|, v, \max(n)) = \min(score)$. Any value larger than $v$ for $p$ would be inconsistent. The upper-cardinality of $p$ is constrained and therefore the filtering of $CoverSize([I_1, \ldots, I_n], \mathcal{D}^+, p)$ based on this upper cardinality is important to prune the search tree. A similar reasoning is used to prune $\min(n)$.

## 5 Experiments

In this section, we report the experimental results on frequent, closed as well as discriminative itemset mining. Each experiment is driven by a concrete question. All experiments were run in the JVM with maximum memory set to 8GB on PCs with Intel Core i5 64bits processor (2.7GHz) and 8GB of RAM running Linux Mint 17.3. Execution time is limited to 1000 seconds.

*Datasets and mining algorithms.* We use data from the FIMI[4] repository and from the CP4IM[5] website. The properties of the datasets are presented in Table 1 (first column) and Table 2a (these latter are labelled positive/negative datasets). We compare with the following methods:

- Frequent Itemset Mining: *FIMCP* [19] using the Gecode solver [17], *DMCP* [30] a custom CP bitvector solver, and four dedicated algorithms namely Borgelt's *Apriori* and *Eclat* implementations [7], *Nonordfp* [36] and *LCMv3*[6] [39].
- Closed Frequent Itemset Mining: *FIMCP*, *DMCP*, Borgelt's Apriori and LCMv3 again, as well as *ClosedPattern* [24] using the or-tools solver [18].
- Discriminative Itemset Mining: *CIMCP* [19] based on Gecode and the specialised algorithm *corrmine* [31].

[3] A related generic technique in CP is *shaving* [26].

[4] http://fimi.ua.ac.be/data/

[5] https://dtai.cs.kuleuven.be/CP4IM/datasets/

[6] http://research.nii.ac.jp/~uno/codes.htm (v3 is fastest of all versions in our experiments)

| tid | itemset |
|---|---|
| 1 | {A,B,D} |
| 2 | {B,C,D} |
| 3 | {A,B,D} |
| 4 | {A,B,C,D} |
| 5 | {B,C} |

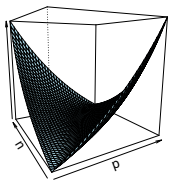
**(a)**

| item | tid-set |
|---|---|
| A | {1,3,4} |
| B | {1,2,3,4,5} |
| C | {2,4,5} |
| D | {1,2,3,4} |

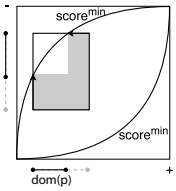
**(b)**

| item | tid-bitvector [1, 2, 3, 4, 5] |
|---|---|
| A | [1 0 1 1 0] |
| B | [1 1 1 1 1] |
| C | [0 1 0 1 1] |
| D | [1 1 1 1 0] |

**(c)**




**(d)**




**(e)**

**Fig. 1: a,b,c)** Three equivalent representations of itemset databases, **d)** $\chi^2$ ZDC function and **e)** Filtering of the $ZDC(|\mathcal{D}^+|, |\mathcal{D}^-|, p, n, score)$ constraint. Note: $c+ = p$, $c- = n$, **a)** Horizontal sparse - $\mathcal{H}$, **b)** Vertical sparse - $\mathcal{V}$ **c)** Vertical dense - $\mathcal{D}$

**Table 1:** CPU runtime for several algorithms vs *CoverSize*.(TO≡TimeOut; * ≡CoverSize+CoverClosure; $\rho \equiv density = \frac{1}{|\mathcal{T}|\times|\mathcal{I}|}\sum_{t\in\mathcal{T},i\in\mathcal{I}}\mathcal{D}_{ti}$)

| Name $\lvert\mathcal{T}\rvert\times\lvert\mathcal{I}\rvert$ $\rho$(%) | $\theta$ | Frequent | | | | | | | | Closed | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | CP-based | | | | Specialized | | | | CP-based | | | | | Specialized | |
| | | *FIMCP* | *DMCP* | *CoverSize-bitset* | *CoverSize* | *Apriori* | *Eclat* | *Nonordfp* | *LCMv3* | *FIMCP* | *ClosedPattern* | *CoverSize-DC** | *CoverSize** | *DMCP* | *Apriori-close* | *LCMv3* |
| retail 88162×16470 ($\rho$=0.06) | 80 | TO | 6.91 | 25.76 | 5.33 | 0.60 | 5.81 | 0.98 | **0.21** | TO | TO | 394.48 | 45.09 | 16.95 | 0.82 | **0.26** |
| | 60 | TO | 10.45 | 33.87 | 7.37 | 0.71 | 8.26 | 1.31 | **0.24** | TO | TO | 952.83 | 67.74 | 25.10 | 1.03 | **0.31** |
| | 40 | TO | 15.96 | 65.13 | 11.19 | 0.77 | 11.42 | 1.83 | **0.27** | TO | TO | TO | 125.67 | 41.78 | 1.29 | **0.49** |
| | 20 | TO | 26.81 | 132.53 | 19.74 | 1.10 | 17.86 | 2.56 | **0.43** | TO | TO | TO | 226.32 | 94.24 | 1.61 | **0.48** |
| | 10 | TO | 40.03 | 191.05 | 37.08 | 1.73 | 24.63 | 3.68 | **0.39** | TO | TO | TO | 366.83 | 238.71 | 2.48 | **0.66** |
| online-retails 541909×2603 ($\rho$=0.1) | 70 | TO | 11.00 | 54.19 | 8.27 | 2.75 | 14.27 | **0.31** | 1.59 | TO | TO | 242.80 | 98.67 | 11.00 | **1.28** | 1.43 |
| | 40 | TO | 11.33 | 59.60 | 8.00 | 4.78 | 15.07 | **0.40** | 1.43 | TO | TO | 497.14 | 111.34 | 12.06 | **1.28** | 1.51 |
| | 10 | TO | 11.49 | 86.66 | 8.49 | 2.15 | 15.61 | **0.43** | 1.51 | TO | TO | 907.68 | 131.94 | 13.31 | **1.36** | 1.52 |
| | 5 | TO | 15.64 | 84.05 | 8.82 | 2.13 | 14.56 | **0.31** | 1.32 | TO | TO | TO | 148.29 | 13.72 | **1.31** | 1.65 |
| | 1 | TO | TO | TO | TO | 2.18 | 15.66 | **0.42** | 1.22 | TO | TO | TO | TO | 14.99 | **1.44** | 1.60 |
| BMSWebView1 59602×497 ($\rho$=0.5) | 48 | TO | 1.92 | 1.51 | 0.69 | 0.08 | 0.51 | 0.11 | **0.03** | TO | TO | 3.10 | 2.54 | 48.04 | 0.29 | **0.11** |
| | 36 | TO | 17.83 | 7.23 | 1.87 | 0.88 | 0.37 | 0.30 | **0.11** | TO | TO | 3.77 | 3.74 | 512.09 | 1.55 | **0.26** |
| | 34 | TO | 63.93 | 23.07 | 8.12 | 7.04 | 0.43 | 0.60 | **0.10** | TO | TO | 3.99 | 4.57 | 746.61 | 10.46 | **0.37** |
| | 32 | TO | TO | TO | TO | TO | 0.68 | 60.63 | **0.28** | TO | TO | 5.12 | 8.24 | TO | TO | **0.47** |
| | 30 | TO | TO | TO | TO | TO | **0.53** | TO | 1.22 | TO | TO | 6.61 | 13.50 | TO | TO | **0.67** |
| T10I4D100K 100000×870 ($\rho$=1.0) | 500 | TO | 1.07 | 3.58 | 0.79 | 0.48 | 3.32 | 0.80 | **0.36** | TO | 8.32 | 8.20 | 7.81 | 1.04 | 0.81 | **0.34** |
| | 400 | TO | 0.98 | 4.1 | 1.03 | 0.49 | 3.74 | 0.58 | **0.29** | TO | 13.06 | 9.27 | 8.88 | 1.12 | 0.67 | **0.43** |
| | 300 | TO | 1.39 | 4.96 | 1.27 | 0.69 | 4.14 | 0.94 | **0.30** | TO | 25.28 | 11.12 | 10.51 | 1.30 | 0.85 | **0.41** |
| | 200 | TO | 2.65 | 6.59 | 1.28 | 0.63 | 3.95 | 0.70 | **0.39** | TO | 77.20 | 12.95 | 10.98 | 1.61 | 1.07 | **0.38** |
| | 100 | TO | 2.35 | 7.27 | 1.78 | 1.02 | 5.08 | 1.01 | **0.53** | TO | 125.26 | 15.79 | 15.07 | 2.11 | 1.15 | **0.58** |
| pumsb-star 49046×2088 ($\rho$=2.0) | 18000 | 302.06 | 1.02 | 1.26 | 0.84 | 6.34 | 0.77 | 0.25 | **0.21** | TO | 56.55 | 0.99 | 0.89 | 2.11 | 5.11 | **0.22** |
| | 16000 | 375.07 | 2.57 | 2.55 | 1.52 | 18.81 | 1.04 | 0.27 | **0.22** | TO | 120.33 | 0.80 | 0.93 | 4.04 | 15.59 | **0.27** |
| | 14000 | 563.21 | 9.14 | 4.72 | 3.36 | 81.93 | 1.34 | 0.31 | **0.26** | TO | 275.23 | 1.65 | 2.25 | 6.22 | 57.99 | **0.30** |
| | 12000 | TO | 33.66 | 20.12 | 10.16 | 285.36 | 1.95 | 0.62 | **0.38** | TO | 601.72 | 4.51 | 5.55 | 14.04 | 284.39 | **0.44** |
| | 10000 | TO | TO | TO | TO | TO | 3.45 | 164.76 | **0.45** | TO | TO | 10.37 | 13.97 | 26.96 | TO | **0.54** |
| pumsb 49046×2113 ($\rho$=3.0) | 40000 | 237.09 | 2.82 | 2.02 | 1.51 | 1.45 | 0.87 | **0.14** | 0.15 | TO | 212.83 | 1.84 | 2.14 | 7.22 | 1.46 | **0.15** |
| | 35000 | 889.08 | 33.87 | 13.20 | 10.26 | 15.92 | 3.36 | 0.21 | **0.20** | TO | TO | 12.71 | 16.76 | 43.48 | 15.98 | **0.27** |
| | 30000 | TO | 220.03 | 124.65 | 63.91 | 356.90 | 10.55 | 0.54 | **0.27** | TO | TO | 62.81 | 82.47 | 121.24 | 370.78 | **0.60** |
| | 25000 | TO | TO | TO | 602.72 | TO | 66.46 | 3.56 | **1.04** | TO | TO | 468.99 | 611.62 | TO | TO | **1.54** |
| accidents 340183×468 ($\rho$=7.0) | 300000 | 50.02 | 0.78 | 0.16 | **0.05** | 1.23 | 1.02 | 0.23 | 0.21 | 221.80 | 0.13 | 0.08 | **0.07** | 0.80 | 1.48 | 0.19 |
| | 250000 | 55.69 | 1.08 | 0.18 | **0.17** | 1.75 | 1.26 | 0.33 | 0.40 | 253.90 | 1.71 | 0.17 | **0.14** | 1.16 | 2.00 | 0.21 |
| | 200000 | 112.55 | 0.99 | 0.34 | **0.33** | 2.46 | 1.76 | 0.35 | 0.62 | 302.97 | 8.57 | **0.56** | 0.68 | 1.41 | 2.55 | 0.56 |
| | 150000 | 386.51 | 2.87 | 1.52 | 1.32 | 17.83 | 3.12 | **0.52** | 1.06 | 575.32 | 61.60 | 5.00 | 5.09 | 3.71 | 18.91 | **0.99** |
| | 100000 | TO | 18.08 | 10.69 | 7.79 | 116.26 | 7.32 | **0.74** | 1.73 | TO | 570.38 | 47.55 | 43.75 | 23.63 | 140.26 | **1.47** |
| mushroom 8124×119 ($\rho$=19.0) | 600 | 33.06 | 0.85 | 2.14 | 1.92 | 14.61 | 0.30 | 0.05 | **0.04** | 8.38 | 0.62 | 0.73 | 0.84 | 0.16 | 10.09 | **0.06** |
| | 400 | 106.71 | 3.48 | 5.52 | 5.43 | 58.46 | 0.29 | 0.09 | **0.04** | 14.86 | 1.08 | 0.70 | 0.67 | 0.20 | 36.11 | **0.11** |
| | 200 | 449.85 | 16.77 | 23.37 | 20.10 | 133.54 | 0.37 | 0.27 | **0.07** | 20.12 | 2.35 | 0.75 | 1.56 | 0.35 | 112.82 | **0.17** |
| | 100 | TO | 67.09 | 96.13 | 68.46 | 264.69 | 0.65 | 0.95 | **0.10** | 27.10 | 4.30 | 1.11 | 2.91 | 0.63 | 284.15 | **0.24** |
| soybeans 630×50 ($\rho$=32.0) | 16 | 1.21 | 0.47 | 1.02 | 1.03 | 0.45 | 0.03 | 0.02 | **0.00** | 0.31 | 0.20 | 0.36 | 0.35 | 0.08 | 0.71 | **0.02** |
| | 13 | 1.57 | 0.70 | 1.40 | 1.44 | 0.44 | 0.03 | 0.02 | **0.01** | 0.32 | 0.25 | 0.32 | 0.28 | 0.12 | 0.95 | **0.02** |
| | 10 | 2.54 | 0.75 | 1.69 | 1.44 | 0.71 | 0.04 | 0.03 | **0.02** | 0.34 | 0.29 | 0.40 | 0.29 | 0.11 | 1.20 | **0.02** |
| | 7 | 3.54 | 1.46 | 2.35 | 2.07 | 0.84 | 0.05 | 0.03 | **0.01** | 0.47 | 0.33 | 0.23 | 0.22 | 0.14 | 1.61 | **0.03** |
| | 4 | 7.05 | 3.86 | 4.02 | 3.87 | 1.69 | 0.05 | 0.07 | **0.02** | 0.42 | 0.42 | 0.25 | 0.20 | 0.18 | 2.98 | **0.03** |
| chess 3196×75 ($\rho$=49.0) | 2000 | 3.71 | 0.30 | 1.59 | 1.39 | 3.14 | 0.11 | 0.01 | **0.01** | 1.85 | 1.71 | 1.11 | 1.02 | 0.42 | 3.48 | **0.09** |
| | 1500 | 46.05 | 3.05 | 4.81 | 3.88 | 77.85 | 0.39 | 0.11 | **0.07** | 14.06 | 15.53 | 4.73 | 3.42 | 1.88 | 69.56 | **0.59** |
| | 1000 | 577.29 | 35.16 | 52.60 | 44.94 | 849.49 | 2.15 | 0.68 | **0.37** | 101.68 | 96.65 | 28.11 | 22.76 | 14.96 | 885.14 | **4.90** |
| | 500 | TO | 959.16 | TO | TO | TO | 19.06 | 12.35 | **2.96** | 882.16 | 900.45 | 304.74 | 282.50 | 144.04 | TO | **51.66** |
| | 250 | TO | TO | TO | TO | TO | 72.69 | 129.05 | **14.42** | TO | TO | TO | TO | 580.64 | TO | **211.99** |

We denote our approach by *CoverSize* and it uses the OscaR solver [33].

**Q1: What is the impact of using a reversible *"sparse"-bitset* over a reversible non-sparse one?** In Table 1 *CoverSize-bitset* is the same implementation as *CoverSize* but using a reversible bitset implementation that does not check for zero words. The results on *Frequent* in Table 1 convincingly show that using the sparse data structure is always better and sometimes an order of magnitude faster, especially on large and sparse datasets.

For closed, we can also compare *Coversize-DC** to *ClosedPattern* [24] which uses the same filtering rules but in a single global constraint and with a different solver and non-reversible non-sparse bitsets [24]. We only have the binaries and though different solvers will perform differently, or-tools has won MiniZinc challenge [38] gold medals and so the remarkable difference in runtime with our method provides strong evidence that the reversible sparse bitset is a well suited and very scalable data structure for itemset propagators.

**Q2: Is domain consistency interesting for closed frequent itemset mining?** In [24] the authors concluded that using the domain consistent version of Rule 6 dominates the simpler non-domain consistent one because of the resulting reduction in number of explored nodes. We reran the same experiment, *CoverSize-DC** and *CoverSize** in Table 1, and the conclusion changes when using reversible sparse bitsets: while on pumsb and pumsb-star the runtime increases when using the simpler non-DC version, on the other datasets it is similar or faster to use the simpler one. For the largest and sparsest datasets retail and online-retails, the difference is even up to an order of magnitude.

**Q3: How does *CoverSize* compare with existing approaches?** The *CoverSize* approach clearly outperforms the decomposition-based *FIMCP*. For frequent, *CoverSize* is on par (sometimes somewhat better or worse) with *DMCP*, the dedicated CP solver which uses bitvector variables. By profiling the execution we observed that for the instances where *DMCP* was faster (such as *mushroom*) only 1% of the time was spent in *CoverSize*. The remaining time is devoted to the solver (propagation management, search, trailing, ...), which a dedicated solver like *DMCP* has less overhead in. Hence, here we show that similar performance can be achieved with a generic solver, through the use of global constraints with carefully designed data structures.

For closed, our approach outperforms *FIMCP*, and also *ClosedPattern* as discussed in Q2. The differences between *CoverSize* and *DMCP* become more varied and pronounced, for example for the sparsest retail and online-retails dataset in favor of DMCP, and for BMSWebView1 in favor of *CoverSize*.

*Specialised algorithms.* There remains a significant gap between CP-based methods and specialised methods though, and especially the highly praised LCMv3 algorithm lives up to its reputation. It should be pointed out that these algorithms do not allow any constraints, and for example a version of LCM (LCMv5) that allow some constraints is also remarkably slower. For denser datasets, our method does typically outperform Apriori.

**Q4: What is the difference in performance for discriminative itemset mining with *CoverSize*?** The state-of-the-art for this problem is the

generic CP-based CIMCP method and the specialized *corrmine* method which implement the same bounds [31]. Table 2b shows a comparison using Information Gain as the ZDC measure, which is the one implemented in *corrmine*. Despite the stronger filtering of CIMCP, *CoverSize* outperforms *CIMCP* for the most difficult instances showing the importance of globals with a good data structure. *corrmine* is superior though specialized to this specific problem.

## 6 Conclusion and Perspectives

We showed that compared to the *ClosedPattern* approach [24] of using a global constraint for frequent closed itemset mining, both generality and efficiency can be significantly improved. Generality can be improved by a separation of concerns in terms of global constraints. We propose to use one global constraint that exposes the frequency through a decision variable which can then be used in other constraints. For example frequency constraints, objective functions or discrimination scores. Another global constraint can be used to enforce the closure property, though care has to be taken when combining it with other constraints.

Efficiency-wise we showed the connection with a well-known constraint that also has to handle a lot of data: the table constraint. Using the *Reversible Sparse bitset* data structure that was recently proposed [14] allows our global constraints to scale to even larger and sparser datasets while still in a generic CP solver. This is relevant not just for frequency-based itemset mining, but also for other existing as well as novel data mining problems in CP, and perhaps beyond.


## Acknowledgments

The research is supported by the FRIA-FNRS (Fonds pour la Formation à la Recherche dans l'Industrie et dans l'Agriculture, Belgium) and FWO (Research Foundation – Flanders). We also thank Willard Zhan for his help with the reduction proof.


**Table 2:** Runtimes, in seconds, for discriminative itemset mining

| ***a)*** *Dataset features.* | | | | ***b)*** *Discriminative* | | | ***a)*** | | | | ***b)*** | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Name | Dense | Trans | Item | CIMCP | CoverSize | corrmine | Name | Dense | Trans | Item | CIMCP | CoverSize | corrmine |
| anneal | 0.45 | 812 | 93 | **0.167** | 0.24 | 0.014 | letter | 0.5 | 20000 | 224 | 54.255 | **4.547** | 0.367 |
| australian-cr | 0.41 | 653 | 125 | **0.166** | 0.195 | 0.012 | mushroom | 0.18 | 8124 | 119 | 15.979 | **0.069** | 0.025 |
| breast-wisc | 0.5 | 683 | 120 | **0.193** | 0.345 | 0.037 | pendigits | 0.5 | 7494 | 216 | 2.939 | **1.196** | 0.138 |
| diabetes | 0.5 | 768 | 112 | **1.564** | 1.769 | 0.28 | primary-t | 0.48 | 336 | 31 | **0.02** | 0.058 | 0.003 |
| german-cr | 0.34 | 1000 | 112 | **1.521** | 1.659 | 0.09 | segment | 0.5 | 2310 | 235 | 1.154 | **0.15** | 0.052 |
| heart-clevel | 0.47 | 296 | 95 | **0.175** | 0.221 | 0.055 | soybean | 0.32 | 630 | 50 | **0.046** | 0.048 | 0.003 |
| hypothyroid | 0.49 | 3247 | 88 | 0.592 | **0.118** | 0.016 | splice-1 | 0.21 | 3190 | 287 | 22.341 | **0.113** | 0.025 |
| ionosphere | 0.5 | 351 | 445 | 1.047 | **0.336** | 0.23 | vehicle | 0.5 | 846 | 252 | 0.551 | **0.55** | 0.094 |
| kr-vs-kp | 0.49 | 3196 | 73 | 0.698 | **0.145** | 0.01 | yeast | 0.49 | 1484 | 89 | 3.386 | **1.366** | 0.818 |
| | | | | | | | *Average. when found* | | | | 5.933 | 0.729 | 0.126 |

## References


1. Aggarwal, C.: An Introduction to Frequent Pattern Mining, pp. 1–17. Springer (2014)
2. Agrawal, R., Imieliński, T., Swami, A.: Mining association rules between sets of items in large databases. International Conference on Management of Data (SIGMOD) 22(2), 207–216 (Jun 1993)
3. Aoga, J.O., Guns, T., Schaus, P.: An efficient algorithm for mining frequent sequence with constraint programming. ECML PKDD 2010 European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases 9853 (2016)
4. Aoga, J.O., Guns, T., Schaus, P.: Mining time-constrained sequential patterns with constraint programming. Constraints pp. 1–23 (2017)
5. Bessiere, C., Régin, J.C.: Arc consistency for general constraint networks: preliminary results. In: International Joint Conference on Artificial Intelligence (IJCAI) (1997)
6. Bonchi, F., Lucchese, C.: On closed constrained frequent pattern mining. In: ICDM '04. Fourth IEEE International Conference on Data Mining. pp. 35–42 (Nov 2004)
7. Borgelt, C.: Efficient implementations of Apriori and Eclat. In: FIMI: Workshop on Frequent Itemset Mining Implementations (2003)
8. Borgelt, C.: Frequent item set mining. Wiley Interdisciplinary Reviews: Data Mining and Knowl. Discovery 2(6), 437–456 (2012)
9. Bringmann, B., Zimmermann, A.: Tree 2–Decision Trees for tree structured data. In: European Conference on Principles of Data Mining and Knowledge Discovery. pp. 46–58. Springer (2005)
10. Bringmann, B., Zimmermann, A., De Raedt, L., Nijssen, S.: Don't be afraid of simpler patterns. In: PKDD. vol. 4213, pp. 55–66. Springer (2006)
11. Cheng, H., Yan, X., Han, J., Philip, S.Y.: Direct discriminative pattern mining for effective classification. In: Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on. pp. 169–178. IEEE (2008)
12. Cheng, K.C., Yap, R.H.: An MDD-based generalized arc consistency algorithm for positive and negative table constraints and some global constraints. Constraints 15(2), 265–304 (2010)
13. De Raedt, L., Guns, T., Nijssen, S.: Constraint programming for itemset mining. In: International Conference on Knowledge Discovery and Data Mining (SIGKDD). pp. 204–212 (2008)
14. Demeulenaere, J., Hartert, R., Lecoutre, C., Perez, G., Perron, L., Régin, J.C., Schaus, P.: Compact-Table: Efficiently Filtering Table Constraints with Reversible Sparse Bit-Sets, pp. 207–223. Springer (2016)
15. Deshpande, M., Kuramochi, M., Wale, N., Karypis, G.: Frequent substructure-based approaches for classifying chemical compounds. IEEE Transactions on Knowledge and Data Engineering 17(8), 1036–1050 (2005)
16. Fan, W., Zhang, K., Cheng, H., Gao, J., Yan, X., Han, J., Yu, P., Verscheure, O.: Direct mining of discriminative and essential frequent patterns via model-based search tree. In: Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 230–238. ACM (2008)
17. Gecode Team: Gecode: Generic constraint development environment (2006), available from http://www.gecode.org
18. Google: Google optimization tools (2015), available from https://developers.google.com/optimization/

19. Guns, T., Nijssen, S., De Raedt, L.: Itemset mining: A constraint programming perspective. Artificial Intelligence 175(12-13), 1951–1983 (2011)
20. Jabbour, S., Sais, L., Salhi, Y.: The top-k frequent closed itemset mining using top-k sat problem. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases. pp. 403–418. Springer (2013)
21. Jabbour, S., Sais, L., Salhi, Y.: Mining top-k motifs with a sat-based framework. Artificial Intelligence (2015)
22. Kemmar, A., Loudni, S., Lebbah, Y., Boizumault, P., Charnois, T.: PREFIX-PROJECTION global constraint for sequential pattern mining. In: International Conference on Principles and Practice of Constraint Programmingn (CP). pp. 226–243. Springer (2015)
23. Knuth, D.: The Art of Computer Programming: Combinatorial Algorithms, vol. 4. Addison-Wesley (2015)
24. Lazaar, N., Lebbah, Y., Loudni, S., Maamar, M., Lemière, V., Bessiere, C., Boizumault, P.: A global constraint for closed frequent pattern mining. In: International Conference on Principles and Practice of Constraint Programmingn (CP). pp. 333–349. Springer (2016)
25. Lecoutre, C.: STR2: optimized simple tabular reduction for table constraints. Constraints 16(4), 341–371 (2011)
26. Lhomme, O.: Quick shaving. In: Proceedings of the 20th national conference on Artificial intelligence-Volume 1. pp. 411–415. AAAI Press (2005)
27. Morishita, S., Sese, J.: Traversing itemset lattice with statistical metric pruning. In: Vianu, V., Gottlob, G. (eds.) Proceedings of the Nineteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, May 15-17, 2000, Dallas, Texas, USA. pp. 226–236. ACM (2000)
28. Negrevergne, B., Dries, A., Guns, T., Nijssen, S.: Dominance programming for itemset mining. In: Data Mining (ICDM), 2013 IEEE 13th International Conference on Data Mining. pp. 557–566. IEEE (2013)
29. Negrevergne, B., Guns, T.: Constraint-based sequence mining using constraint programming. In: CPAIOR 2015. pp. 288–305. Springer (2015)
30. Nijssen, S., Guns, T.: Integrating constraint programming and itemset mining. In: ECML PKDD 2010 European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases. pp. 467–482 (2010)
31. Nijssen, S., Guns, T., De Raedt, L.: Correlated itemset mining in ROC space: a constraint programming approach. In: International Conference on Knowledge Discovery and Data Mining (SIGKDD). pp. 647–656. ACM (2009)
32. Nijssen, S., Zimmermann, A.: Constraint-Based Pattern Mining, pp. 147–163. Springer (2014)
33. OscaR Team: OscaR: Scala in OR (2012), available from `https://bitbucket.org/oscarlib/oscar`
34. Pasquier, N., Bastide, Y., Taouil, R., Lakhal, L.: Discovering frequent closed itemsets for association rules. In: Beeri, C., Buneman, P. (eds.) Database Theory - ICDT '99, 7th International Conference, Jerusalem, Israel, January 10-12, 1999, Proceedings. Lecture Notes in Computer Science, vol. 1540, pp. 398–416. Springer (1999)
35. Perez, G., Régin, J.C.: Improving GAC-4 for table and MDD constraints. In: International Conference on Principles and Practice of Constraint Programmingn (CP). pp. 606–621. Springer (2014)
36. Rácz, B.: nonordfp: An FP-growth variation without rebuilding the FP-tree. In: FIMI: Workshop on Frequent Itemset Mining Implementations (2004)

37. de Saint-Marcq, V.l.C., Schaus, P., Solnon, C., Lecoutre, C.: Sparse-sets for domain implementation. In: CP workshop on - Techniques foR Implementing Constraint programming Systems (TRICS). pp. 1–10 (2013)
38. Stuckey, P.J., Becket, R., Fischer, J.: Philosophy of the minizinc challenge. Constraints 15(3), 307–316 (2010)
39. Uno, T., Kiyomi, M., Arimura, H.: LCM Ver.3: Collaboration of array, bitmap and prefix tree for frequent itemset mining. In: Proceedings of the 1st international workshop on open source data mining: frequent pattern mining implementations (OSDM '05). pp. 77–86. OSDM '05, ACM (2005)