k-Pattern Set Mining under Constraints

Tias Guns, Siegfried Nijssen, Luc De Raedt

Abstract—We introduce the problem of k-pattern set mining, concerned with finding a set of k related patterns under constraints. This contrasts to regular pattern mining, where one searches for many individual patterns. The k-pattern set mining problem is a very general problem that can be instantiated to a wide variety of well-known mining tasks including concept-learning, rule-learning, redescription mining, conceptual clustering and tiling. To this end, we formulate a large number of constraints for use in k-pattern set mining, both at the local level, that is, on individual patterns, and on the global level, that is, on the overall pattern set.
Building general solvers for the pattern set mining problem remains a challenge. Here, we investigate to what extent constraint programming (CP) can be used as a general solution strategy. We present a mapping of pattern set constraints to constraints currently available in CP. This allows us to investigate a large number of settings within a unified framework and to gain insight in the possibilities and limitations of these solvers. This is important as it allows us to create guidelines in how to model new problems successfully and how to model existing problems more efficiently. It also opens up the way for other solver technologies.

Index Terms—Data Mining, Pattern Set Mining, Constraints, Constraint Programming.

1 Introduction

The problem of *local* pattern mining can be formalised as that of finding the set of patterns $\text{Th}(\mathcal{L}, p, \mathcal{D}) = \{\pi \in \mathcal{L} | p(\pi, \mathcal{D}) \text{ is true}\}$, that is, the set of all patterns $\pi \in \mathcal{L}$ that satisfy a constraint p with respect to a database $\mathcal{D}$. Numerous approaches to pattern mining have been developed to effectively find the patterns adhering to a set of constraints; a well-known example is the problem of finding frequent itemsets.

In recent years it is increasingly recognized that the result of a pattern mining operation can almost never be used directly and needs to be post-processed in order to become useful. The reasons are that the generated set of patterns is typically too large, that is, the set of patterns is too hard to interpret as the patterns interact with one another. This has led to a typical *step-wise* procedure in which pattern mining only forms an *intermediate* step in the knowledge discovery process. In the first step, the patterns adhering to some constraints are exhaustively searched for. These patterns are usually called *local* patterns. In the second step, some patterns are selected and combined in a heuristic way to create a *global* model. One example is associative classification, where systems such as CBA [1] and CMAR [2] build a classifier from a set of association rules; other examples are concept-learning [3], conceptual clustering [4], redescription mining [5] and tiling [6]. These algorithms, which calculate a small set of patterns, are algorithms that solve one particular instance of the *pattern set mining problem*.

In contrast to constraint-based local pattern mining, where mining problems are often well formalized and anti-monotonicity is one of the guiding principles for the development of algorithms, no unifying principles or algorithms exist for pattern set mining. Pattern set mining tasks have been solved using wide ranges of heuristics and greedy search strategies, often by searching for patterns in a first step and by filtering these patterns in a second step. The main problem with these approaches is that it is unclear how they extend towards new and unsolved problems. Indeed, to the best of the authors' knowledge, there currently is no general approach for finding pattern sets that generalizes over a large number of settings.

In this article, we make a first step towards formally specifying pattern set mining problems and solving them by means of general algorithms. We develop a framework in which a multitude of tasks, including concept-learning, conceptual clustering, redescription mining and tiling, can be formalized. The main idea in this framework is to formalize mining tasks as problems of finding k patterns that together satisfy constraints. In contrast to earlier approaches, where constraints are typically only formalized on the local level, that is, on individual patterns, within this framework we also formalize constraints on the global level, that is on the pattern set as a whole. Both levels of constraints are formalized at the same time, that is, in a single specification; we present a high-level modelling language, independent from underlying frameworks, and show how to use it to formulate many well-known tasks. A key feature is hence that we open up the possibility that mining problems are not solved in multiple steps, but also in one single step.

Our vision is that these declaratively specified problems are solved using a general solver in a uniform way. Developing such solvers is a challenge. In this article, we take one constraint solving technology as starting point, *constraint programming*, and investigate to what extent it can be used to solve the k-pattern set mining problem. Constraint programming is a generic framework for

The authors are with the Department of Computer Science, K.U. Leuven, Belgium.
Email: {firstname.lastname}@cs.kuleuven.be

solving combinatorial and optimization problems under constraints. It has been used successfully in numerous applications, including constraint-based mining of individual patterns [7]. The key power of CP lies in its generic approach to problem solving: users model a problem by specifying constraints, and the CP solver will use those constraints to find the solutions. This has the advantage that new problems can be solved by only changing the specification in terms of constraints; a new solver is not needed. We will use these advantages of CP to demonstrate the possibilities of a declarative approach to data mining.

A potential issue is that CP solves problems using exhaustive search. In our case, a CP system will also attempt to solve the mining problem in one exhaustive search step. Most current approaches to pattern set mining do not use one exhaustive search step; we can hence not expect that this approach is feasible in all cases. Thus, a main task is to identify the limitations of this approach. We do this by considering the following questions:

- which pattern set mining tasks can be solved in reasonable time, using existing CP technology?
- which modeling choices can help to more effectively solve pattern set mining problems?

A major element in answering these questions is the identification of suitable modeling primitives available in today's CP systems. A first set of such guidelines is developed in this article: starting from a set of well-known pattern set mining problems we put constraints in categories that are increasingly harder to solve. This contributes new insights into the relation between local and global constraints, as well as provides directions for modelling new problems. Furthermore, such insights could also be used for developing more heuristic declarative approaches, for instance, when combining local search with exhaustive search. Hence, this work may contribute to the longer term vision of developing general purpose declarative data mining tools.

2 k-Pattern Set Mining

In general, pattern mining is concerned with finding all patterns π that adhere to some constraint p. The patterns are defined by a pattern language $\mathcal{L}$. The constraint p is typically a conjunction of multiple constraints, that can be defined on data $\mathcal{D}$:

$$\mathrm{Th}(\mathcal{L}, p, \mathcal{D}) = \{\pi \in \mathcal{L} \mid p(\pi, \mathcal{D}) \text{ is true}\}. \quad (1)$$

In the prototypical example of itemset mining, we start from an itemset database, that is, a set $\mathcal{D} \subseteq \mathcal{T} \times \mathcal{I}$ where $\mathcal{I}$ is a set of items and $\mathcal{T}$ is a set of transaction identifiers. The traditional itemset mining problem is that of finding $\mathrm{Th}(\mathcal{I}, p, \mathcal{D}) = \{I \subseteq \mathcal{I} \mid p(I, \mathcal{D}) \text{ is true}\}$. The pattern language is here the space of itemsets $2^{\mathcal{I}}$, and p specifies the *local* constraints. The well-known frequent itemset mining problem can be cast within this framework by defining

$$p(I, \mathcal{D}) = true \text{ iff } |\varphi(I)| \geq \theta \quad (2)$$

where $\varphi(I)$ denotes the transactions that contain itemset I, that is,

$$\varphi(I) = \{t \in \mathcal{T} \mid \forall i \in I : (t, i) \in \mathcal{D}\}. \quad (3)$$

One of the problems with local pattern mining is that if the constraint p is not restrictive enough, the set of patterns in $\mathrm{Th}(\mathcal{L}, p, \mathcal{D})$ becomes too large and needs further processing before it can be used. This resulted in the adoption of *two step* procedures in order to arrive at a useful set of patterns. First, all patterns that adhere to some chosen *local constraints* are mined exhaustively. Then, these patterns are combined under a set of *global constraints*, often including an optimisation function f. Because of the size of the local pattern set, usually heuristic techniques are used when searching for the approximately best pattern set. Here we wish to avoid this two step procedure by formulating all the constraints directly on the entire pattern set, that is, on a set containing a fixed number k of patterns.

The problem of *k-pattern set mining under constraints* can now be defined as finding:

$$\mathrm{Th}(\mathcal{L}, p, \mathcal{D}) = \{\Pi \in \mathcal{L}^k \mid p(\Pi, \mathcal{D}) \text{ is true}\}. \quad (4)$$

The pattern set Π consists of k patterns π. In its whole, the pattern set Π forms a global model. The constraint p specifies both local and global constraints at the overall pattern set level. In addition, as the number of pattern sets can become very large, we will study how to find the best pattern set with respect to an optimisation criterion $f(\Pi)$. That is, we study how to find the optimal pattern set Π by searching, for example, for the one with maximum $f(\Pi)$:

$$\arg \max_{\Pi \in \mathrm{Th}(\mathcal{L}, p, \mathcal{D})} f(\Pi), \quad (5)$$

where we still have the possibility to impose a set of constraints in p.

In the rest of this paper, we assume that a k-pattern set Π consists of k individual itemset patterns π. Every pattern π is represented by its itemset I and transaction set T: $\pi = (I, T)$. We write $\Pi = (\pi^1, \ldots, \pi^k) = ((I^1, T^1), \ldots, (I^k, T^k))$. The transaction set consists of all transactions or examples that are covered by the itemset.

2.1 Families of Constraints

In this section we will present four families of constraints. In the next section, we will then show how k-pattern set mining problems can be specified as combinations of constraints from these families. The first family is the family of *individual pattern constraints*. The typical local pattern mining constraints fall into this category. Second, *redundancy constraints* can be used to constrain or minimise the redundancy between different patterns. *Coverage constraints* deal with defining and measuring how well a pattern set covers the data. Given labelled data, the *discriminative constraints* can be used to measure and optimise how well a pattern or pattern set discriminates between positive and negative examples.

We will use the notation $constraint \Leftrightarrow equation$ to define a constraint in terms of an equation consisting of operations on the itemsets I and transaction sets T.

2.1.1 Individual Pattern Constraints

These are constraints that have been identified in the framework of constraint-based pattern mining [8]. We will review some of the most important constraints for the itemset mining problem.

Coverage constraint. Because every itemset I uniquely defines a transaction set $\varphi(I)$, we will explicitly constrain the set of transactions in a pattern $\pi = (I, T)$ to those transactions containing the itemset:

$$coverage(\pi) \Leftrightarrow (T = \varphi(I) = \{t \in \mathcal{T} \mid \forall i \in I : (t, i) \in \mathcal{D}\}). \quad (6)$$

Disjunctive coverage constraint. Alternatively, we can define that a transaction is covered if it is covered by at least one item. This is useful when mining for a disjunction of items.

$$disjcoverage(\pi) \Leftrightarrow T = \{t \in \mathcal{T} \mid \exists i \in I : (t, i) \in \mathcal{D}\}. \quad (7)$$

Closedness constraint. Dually to the *coverage constraint*, we can enforce that an itemset must be the largest itemset that is contained in all selected transactions:

$$closed(\pi) \Leftrightarrow (I = \psi(T) = \{i \in \mathcal{I} \mid \forall t \in T : (t, i) \in \mathcal{D}\}). \quad (8)$$

The two constraints combined define a closure operator $I = \psi(\varphi(I))$. This provides a way to remove redundancy among the patterns found.

Size constraints. The size of a pattern $\pi = (I, T)$ can straightforwardly be measured as $size(\pi) = |I|$. In the general case, we can define a lower or upper bound constraint on any measure. With $\lessgtr$ we will denote any comparison $\lessgtr \in \{<, \leqslant, >, \geqslant, =, \neq\}$. A constraint on the size of the pattern can thus be formulated as

$$size(\pi) \lessgtr \theta \Leftrightarrow |I| \lessgtr \theta. \quad (9)$$

Frequency constraint. The frequency of an itemset is simply the size of its transaction set: $freq(\pi) = |T|$, which can also be constrained as $|T| \lessgtr \theta$.

$$freq(\pi) \lessgtr \theta \Leftrightarrow |T| \lessgtr \theta. \quad (10)$$

2.1.2 Redundancy Constraints

As pointed out in the introduction, an important problem in local pattern mining is that of redundancy among patterns. We already defined the *closedness* constraint that can be used to remove a certain kind of redundancy of individual patterns. However a pattern can still be considered logically redundant if it covers approximately the same transactions as another pattern. One way to measure this redundancy between two patterns is by measuring the similarity or distance between their transaction sets.

Distance measures. We can measure the overlap between two patterns as the size of the intersection between the transaction sets. Likewise, the distinctness of two patterns can be measured by the size of the symmetric difference between the transaction sets.

$$overlap(\pi^1, \pi^2) = \left|T^1 \cap T^2\right|, \quad (11)$$

$$distinct(\pi^1, \pi^2) = \left|(T^1 \cup T^2) \setminus (T^1 \cap T^2)\right|. \quad (12)$$

The distance between two patterns can also be measured using any distance measure between the two transaction sets, for example the Jaccard similarity coefficient or the Dice coefficient.

$$jaccard(\pi^1, \pi^2) = (|T^1 \cap T^2|)/(|T^1 \cup T^2|), \quad (13)$$

$$dice(\pi^1, \pi^2) = (2 * |T^1 \cap T^2|)/(|T^1| + |T^2|). \quad (14)$$

Such measures can be constrained by a comparison operator $\lessgtr \in \{<, \leqslant, >, \geqslant, =, \neq\}$ and a threshold θ.

Combining measures. Since the measures only indicate the redundancy between two patterns, and not an entire pattern set, we often need to aggregate over all pairwise combinations of patterns. A first approach is to constrain the sum of all pairwise evaluations. Using the function name *dist()* to indicate any distance measure, we may define that:

$$sumdist(\Pi) \lessgtr \theta \Leftrightarrow \sum_{i=1}^{k} \sum_{j=i+1}^{k} dist(\pi^i, \pi^j) \lessgtr \theta. \quad (15)$$

An alternative approach is to bound the minimum or maximum value over all evaluations:
$min\left(dist(\pi^1, \pi^2), dist(\pi^1, \pi^3), \ldots, dist(\pi^2, \pi^3), \ldots\right) \lessgtr \theta$.
In the case of upper-bounding the minimum or lower-bounding the maximum, this is equal to constraining every pairwise evaluation separately. For example, when constraining the minimum to be greater than a value θ, this can be rewritten as follows:

$$\min_{i<j} \left(dist(\pi^i, \pi^j)\right) \geq \theta \Leftrightarrow$$
$$dist(\pi^1, \pi^2) \geq \theta, dist(\pi^1, \pi^3) \geq \theta, \ldots \quad (16)$$

2.1.3 Coverage Constraints

Each individual pattern $\pi = (I, T)$ in our setting consists of an itemset and its corresponding transaction set T, because of the above defined *coverage constraint*. The cover of the entire pattern set Π is not explicit in our formulation, but can be deduced from the covers of all the individual patterns.

Pattern set cover. A pattern set can be interpreted as a disjunction of the individual patterns. Hence, we calculate the transaction set of the entire pattern set by taking the union over the individual transaction sets:

$$cover(\Pi) = T^{\Pi} = T^1 \cup \cdots \cup T^k. \quad (17)$$

Frequency of pattern set. The frequency of the pattern set is then calculated exactly like the frequency of an individual pattern: $freq(\Pi) = |T^{\Pi}|$. This can again be constrained:

$$freq(\Pi) \lessgtr \theta \Leftrightarrow |T^{\Pi}| \lessgtr \theta. \quad (18)$$

Area of pattern set. The area of a pattern set was studied in the context of large tile mining [6]. The tile of a pattern

contains all tuples $(t,i) \in \mathcal{D}$ that are covered by the pattern: $tile(\pi) = \{(t,i) \mid t \in T, i \in I\}$. These tuples form a tile or rectangle of 1's in the binary database D. The area of a single pattern is the number of tuples that are covered in the tile: $area(\pi) = |tile(\pi)| = |I| \cdot |T|$. The area of a pattern set can now be defined as the union of all the tiles of the individual patterns. Note that tiles can be overlapping, but every tuple (t,i) covered is only counted once.

$$area(\Pi) = \left|tile(\pi^1) \cup \ldots \cup tile(\pi^k)\right|. \tag{19}$$

2.1.4 Discriminative Constraints

In many cases, one is interested in finding patterns in labelled data, that is, data in which there is a label l attached to each transaction t. We will only consider the case where the label is either positive $(+)$ or negative $(-)$, though this can be extended to more classes. In this setting, the data can be divided into two partitions: the set of transactions $\mathcal{T}^+$ having label $+$, and the transactions $\mathcal{T}^-$ having label $-$. The positive cover of a pattern is the cover of the pattern on the positive examples: $cover^+(\pi) = T \cap \mathcal{T}^+$. Similarly, the negative cover is $cover^-(\pi) = T \cap \mathcal{T}^-$. To simplify our formulas, we will often abbreviate $|cover^+(\pi)|$ by p and $|cover^-(\pi)|$ by n in this section. The same holds for the positive and negative cover of the entire pattern set, where T^Π is defined as in equation (17):

$$cover^+(\Pi) = T^\Pi \cap \mathcal{T}^+, \tag{20}$$
$$cover^-(\Pi) = T^\Pi \cap \mathcal{T}^-. \tag{21}$$

We can also define the closedness constraint only on the positive transactions: $closed^+(\pi) \Leftrightarrow I = \psi(T \cap \mathcal{T}^+)$; similarly for closedness on the negative transactions.

Discriminative measures are typically defined by comparing the number of positive examples p and negative examples n covered to the total number of positives examples P and negatives examples N. The total number of positives P is simply the size of $\mathcal{T}^+$: $P = |\mathcal{T}^+|$, likewise the total number of negatives N is $N = |\mathcal{T}^-|$. The number of positive/negative examples covered by an entire pattern set is then calculated as:

$$p = freq^+(\Pi) = |cover^+(\Pi)| = |T^\Pi \cap \mathcal{T}^+|, \tag{22}$$
$$n = freq^-(\Pi) = |cover^-(\Pi)| = |T^\Pi \cap \mathcal{T}^-|. \tag{23}$$

Accuracy of a pattern set. Accuracy is defined as the proportion of examples that are correctly covered by the pattern set: $(p + (N - n))/(P + N)$. As P and N are constant for a given dataset, one can equivalently optimize the formula $p - n$ [9]. Using equations (22) and (23) and the principles explained above we get the following formulation:

$$accuracy\left(freq^+(\Pi), freq^-(\Pi)\right) = freq^+(\Pi) - freq^-(\Pi) \tag{24}$$

Using the same principles we can also use other discriminatory measures, like weighted accuracy or the Laplace estimate:

$$w\_accuracy : \frac{freq^+(\Pi)}{|\mathcal{T}^+|} - \frac{freq^-(\Pi)}{|\mathcal{T}^-|}, \tag{25}$$

$$Laplace : \frac{freq^+(\Pi) + 1}{freq^+(\Pi) + freq^-(\Pi) + 2}. \tag{26}$$

The topic of identifying such measures has already been studied extensively in pattern mining [10], [1] and in rule learning [9].

2.2 Instantiations

As argued in the introduction, the k-pattern set mining problem is very general and flexible. One of the contributions of this paper is that we show how it can be instantiated to address several well-known data mining problems. This is explained in the following paragraphs. We will present both satisfaction problems and optimisation problems.

Satisfaction problems are specified by listing all the constraints needing to be satisfied. Optimisation problems additionally start with the $\mathrm{maximise}$ or $\mathrm{minimise}$ keywords, indicating the function to optimise. A solution needs to satisfy all constraints; in case of an optimisation problem, only the solution with the highest, or lowest, value for the given optimisation function is sought.

2.2.1 $k-$term DNF Learning and Concept Learning

The main aim when learning a $k-$term formula in disjunctive normal form (DNF) is to learn a formula which performs a binary prediction task as accurately as possible, given a set of labelled training examples. A formal definition of $k-$term DNF learning was given in [3]. Within our framework, we can interpret each clause as an itemset, while the pattern set corresponds to a disjunction of such clauses. We can formalise this problem as finding pattern sets Π satisfying:

$$\begin{aligned} &\forall \pi \in \Pi : && coverage(\pi), \\ &\forall \pi \in \Pi : && closed^+(\pi), \\ & && accuracy\left(freq^+(\Pi), freq^-(\Pi)\right) \geq \theta. \end{aligned} \tag{27}$$

where we choose $\theta = |\mathcal{T}^+|$ if we do not wish to allow for errors on the training data (pure DNF learning). Note that we have added a closedness constraint on the positive transactions; the main motivation for this choice is that for any $k-$term DNF containing arbitrary itemsets, there is an equally good or better $k-$term DNF with only closed-on-the-positive itemsets [11]. Adding this constraint also reduces the space of hypotheses and hence reduces the practical complexity of the problem.

The above formulation would result in all k-pattern sets that are accurate enough. In the concept learning setting we are usually interested only in discovering the most accurate pattern set. To achieve this, the above satisfaction problem is turned into an optimisation problem:

$$\begin{aligned} &\underset{\Pi}{\mathrm{maximise}} && accuracy\left(freq^+(\Pi), freq^-(\Pi)\right), \\ &\forall \pi \in \Pi : && coverage(\pi), closed^+(\pi). \end{aligned}$$

We can replace accuracy with any other discriminative constraint as explained in section 2.1.4. Note that it is easy to add other constraints to this formulation, like a minimum frequency constraint on every individual pattern:

$$\begin{aligned} \underset{\Pi}{\text{maximise}} \quad & \mathit{accuracy}\left(\mathit{freq}^+(\Pi), \mathit{freq}^-(\Pi)\right), \\ \forall \pi \in \Pi: \quad & \mathit{coverage}(\pi), \mathit{closed}^+(\pi), \\ \forall \pi \in \Pi: \quad & \mathit{freq}(\pi) \geq \theta. \end{aligned}$$

In all cases, the result will be a k-pattern set with high accuracy on the examples. Every pattern π will represents a learned concept.

2.2.2 Conceptual Clustering

The main aim of clustering algorithms is to find groups of examples which are similar to each other. In conceptual clustering, the additional goal is to learn a conceptual description for each of these clusters [4]. In this section, we consider a simplified version of conceptual clustering, in which we call two examples similar if they contain the same pattern. Hence, each cluster is described by a pattern, and all examples that are covered by the pattern are part of the cluster. We then formalise conceptual clustering as finding pattern sets Π that do not overlap and cover all the examples:

$$\begin{aligned} \forall \pi \in \Pi: \quad & \mathit{coverage}(\pi), \mathit{closed}(\pi), \\ & \mathit{cover}(\Pi) = \mathcal{T}, \\ \forall \pi^a, \pi^b \in \Pi: \quad & \mathit{overlap}(\pi^a, \pi^b) = 0. \end{aligned}$$

The solutions to this model form the restricted set of clusterings that do not overlap. Still, finding all such clusterings may not be desirable and finding one clustering could be sufficient. In this case, it could be more interesting to search for the best non-overlapping clustering. There are multiple ways to define what the 'best' clustering is. One possible way is to prefer solutions in which the sizes of the clusters do not differ too much from each other. We can formalize this in several possible ways. For example, we could search for solutions in which the minimum size of the clusters is as large as possible:

$$\begin{aligned} \underset{\Pi}{\text{maximise}} \quad & \min\left(\mathit{freq}(\pi^1), \ldots, \mathit{freq}(\pi^k)\right), \\ \forall \pi \in \Pi: \quad & \mathit{coverage}(\pi), \mathit{closed}(\pi), \\ & \mathit{cover}(\Pi) = \mathcal{T}, \\ \forall \pi^a, \pi^b \in \Pi: \quad & \mathit{overlap}(\pi^a, \pi^b) = 0. \end{aligned}$$

The minimum cluster size would be maximal if all clusters have the same size; hence this formulation will prefer more balanced solutions. Alternatively, one could also use the following optimization criterion:

$$\underset{\Pi}{\text{minimise}} \quad \max\left(\mathit{freq}(\pi^1), \ldots\right) - \min\left(\mathit{freq}(\pi^1), \ldots\right);$$

this would enforce a small difference between cluster sizes more directly. We will compare these two settings in the experimental section.

In the general case, other settings may also seem desirable. For instance, the constraint that clusters are not allowed to overlap might seem too restrictive, and one could choose to use the following optimisation criterion:

$$\underset{\Pi}{\text{minimise}} \sum_{\pi^a, \pi^b \in \Pi} \mathit{overlap}(\pi^a, \pi^b),$$

however, we determined experimentally that in many datasets a set of k non-overlapping patterns can be found, and hence this optimization criterion would give the same solution as when using the overlap constraint; further extensions of the model are needed in order to make the setting more useful in practice. Of most interest is probably the incorporation of arbitrary distance functions in the optimisation, as is common in clustering. We here restrict ourselves to distance functions that can be defined over the item and transaction sets of the patterns.

2.2.3 $k-$Tiling

The main aim of tiling [6] is to cover as many 1s in a binary matrix with a given number of patterns or tiles. A tiling can be considered useful as the patterns in a tiling are in some way most characteristic for the data. We can formalise this problem as follows:

$$\begin{aligned} \underset{\Pi}{\text{maximise}} \quad & \mathit{area}(\Pi), \\ \forall \pi \in \Pi: \quad & \mathit{coverage}(\pi), \mathit{closed}(\pi). \end{aligned}$$

The closedness constraint is not strictly necessary, but closed itemsets cover a larger area than their non-closed counterparts.

2.2.4 Redescription Mining

The main aim of redescription mining [5] is to find sets of syntactically different formulas that all cover the same set of transactions; such sets of formulas are of interest as they point towards equivalences in the attributes in the data. We assume given a number of disjoint partitions of items $\mathcal{I}^1 \ldots \mathcal{I}^k$ where $\forall p: \mathcal{I}^p \subseteq \mathcal{I}$ and $\forall p, q, p \neq q: \mathcal{I}^p \cap \mathcal{I}^q = \emptyset$ and can formalize the problem in multiple alternative ways.

We will restrict ourself to finding conjunctive formulas that form redescriptions. In this setting, we may search for a pattern set of size k as follows:

$$\begin{aligned} \underset{\Pi}{\text{maximise}} \quad & \mathit{freq}(\pi^1), \\ \forall \pi \in \Pi: \quad & I \subseteq \mathcal{I}^{p(\pi)}, \\ \forall \pi \in \Pi: \quad & \mathit{covered}(\pi), \mathit{closed}(\pi), \\ \forall \pi^a, \pi^b \in \Pi: \quad & T^a = T^b, \end{aligned}$$

where $p(\pi)$ is the number of the item partition corresponding to that pattern. In the case above, only the frequency of one pattern needs to be maximised as all patterns have to cover exactly the same transactions. As this may be a too strict requirement, one could also solve

the following problem where we search for an accurate redescription of a sufficiently large number of examples:

$$\begin{aligned} \underset{\Pi}{\text{minimise}} \quad & sumdist(\Pi), \\ \forall \pi \in \Pi: \quad & I \subseteq \mathcal{I}^{p(\pi)}, \\ \forall \pi \in \Pi: \quad & covered(\pi), closed(\pi), \\ \forall \pi \in \Pi: \quad & freq(\pi) \geq \theta. \end{aligned}$$

In principle every distance measure from Section 2.1.2 can be used. We compare these two settings further in the experimental section.

3 Constraint Programming

We will show how constraint programming (CP) provides a natural framework for solving k-pattern set mining problems. We review the principles of constraint programming in this section. The next section shows how the constraints of Section 2 can be expressed in a constraint programming system and how the system uses them to find solutions.

Constraint programming addresses combinatorial (optimisation) problems through decomposition and separation of concerns. It separates the modelling of the problem from the solving of the problem: constraint programming starts from a model and the solver searches for a solution.

A *constraint satisfaction problem* (CSP) $(\mathcal{V}, D, \mathcal{C})$ is specified using 1) a finite set of variables $\mathcal{V}$; 2) an initial domain D, which maps every variable $v \in \mathcal{V}$ to a finite set of possible values $D(v)$; and 3) a finite set of constraints $\mathcal{C}$, each a boolean function on a subset of $\mathcal{V}$.

A *constraint optimisation problem* is a CSP $(\mathcal{V}, D, \mathcal{C})$ augmented with an objective function that maps a subset of $\mathcal{V}$ to an evaluation score v_s. The problem is then to find the solution that satisfies all constraints and is maximal (or minimal) with respect to the objective function.

Example 1 (CSP): A family of 5, consisting of a mother, a father, a grandfather and two children has won a holiday for 3 people. The parents decide that at least one of them should join, but the father will only join iff either the mother or the grandfather, but not both, goes. We can model this problem as a CSP by having a variable for every family member, namely G (grandfather), F (father), M (mother) and Ch_1 and Ch_2 (the children). If a person joins, the corresponding variable has value 1 and 0 otherwise; the domain of every variable is $\{0, 1\}$. This is declaratively specified in line (28) below.

$$D(G), D(F), D(M), D(Ch_1), D(Ch_2) = \{0, 1\} \quad (28)$$
$$F + M \geq 1 \quad (29)$$
$$F \leftrightarrow M + G = 1 \quad (30)$$
$$G + F + M + Ch_1 + Ch_2 = 3 \quad (31)$$

We will specify the constraints by summing over these boolean variables. In line (31) above, we specify that only 3 people can go on the holiday, by constraining the sum of all the variables to be equal to 3. Line (29) specifies that at least one of the parents has to join. Finally line (30) specifies that the father joins iff either the mother or grandfather joins, using the shorthand notation that F is true iff $F \neq 0$.

Algorithm 1 Constraint-Search(D)

```
1:  D :=propagate(D)
2:  if D has failed then
3:      return
4:  end if
5:  if ∃v ∈ V : |D(v)| > 1 then
6:      v := arg min_{v∈V, D(v)>1} f(v)
7:      for all d ∈ D(v) do
8:          Constraint-Search(D ∪ {v ↦ {d}})
9:      end for
10: else
11:     Output solution
12: end if
```

In the above example, we had to make some modelling decisions. For example, we could have specified the constraint on line (29) using the boolean formulation $F \vee M$. Also, some languages allow the use of set variables that could model the family as a single set; however this would complicate posting constraints on subsets or single elements such as on lines (29, 30). When modelling a problem it is often possible to specify it in different ways. As different constraints are implemented differently, the choice made can have an effect on runtime efficiency and should be made in that context.

After specifying the CSP, the solver uses the constraints in its search for solution(s). The search tree of a CSP is ordered from general to specific domains. The root node consists of the initial domain D containing all the possible values of each variable. Solutions are found in the leaves of the search tree, where every variable v has only one value in its domain $D(v)$. A branch in the search tree is created by assigning a value to a variable.

There are several search strategies that can be applied. An outline of a general depth-first search algorithm is given in Algorithm 1. The constraints are propagated in line 1. If a constraint is violated (line 2) the search will backtrack. If not all variables are assigned (line 5), the search algorithm will choose a variable (line 6) and branch over each of the values (line 7). For optimisation problems, the above algorithm can easily be changed into a branch-and-bound algorithm. In this case a constraint is added on the evaluation score v_s (line 11); this constraint is updated each time a better solution is found than the currently best known one, and hence enforces that solutions worse than the currently best known one are ignored.

Propagation of constraints is the essential operation in constraint programming: it is the act of removing a value from the domain of a variable when it can be determined that the value can no longer be part of any viable solution. Propagation is realised through propagators; every

Algorithm 2 Propagator for $\sum B \geq \theta$

1: $Lowerbound := \sum_{B_i \in B} \min D(B_x)$
2: $Upperbound := \sum_{B_i \in B} \max D(B_x)$
3: **if** $Lowerbound \geq \theta$ **then**
4: % Constraint is respected
5: **end if**
6: **if** $Upperbound < \theta$ **then**
7: % Constraint is violated
8: **end if**
9: **if** $Upperbound = \theta$ **then**
10: for all $B_i \in B : D(B_i) = \{\max D(B_i)\}$ % Propagate
11: **end if**

constraint is implemented by a propagator. The constraints we will use are often of the form $Function(V) \lesseqgtr X$ where $\lesseqgtr \in \{<, \leqslant, >, \geqslant, =, \neq\}$, V is a set of variables and X is either another variable or a constant. We shall use so-called *bound-consistent* propagators, which operate on the bounds of the variables. In such propagators, the upper- and lower-bound of the variables are used and possibly constrained. The upper-bound of a variable V is the largest value in its domain, which we denote by $Upperbound(V) = \max D(V)$; likewise the lower-bound is the minimal value in the domain. The upper- and lower-bounds of variables in a set V can often be used to calculate an upper- and lower-bound on the outcomes of a function $f(V)$. This is illustrated in the table below for basic arithmetic functions over 2 variables:

Function	Lower-bound	Upper-bound
$V_1 + V_2$	$\min D(V_1) + \min D(V_2)$	$\max D(V_1) + \max D(V_2)$
$V_1 - V_2$	$\min D(V_1) - \max D(V_2)$	$\max D(V_1) - \min D(V_2)$
$V_1 * V_2$	$\min D(V_1) * \min D(V_2)$	$\max D(V_1) * \max D(V_2)$
$V_1 \div V_2$	$\min D(V_1) \div \max D(V_2)$	$\max D(V_1) \div \min D(V_2)$

These bounds can be used in several ways to update the domains of variables. In a constraint of the kind $Function(V) \lesseqgtr X$ where X is a variable, we can update the bounds of the variable X. Furthermore we can iteratively fix variables in V (keeping the domains of other variables unchanged); those values can be removed from consideration which result in a bound that no longer satisfies the constraint.

As an example, consider a vector of boolean variables $B = (B_1, B_2, \ldots, B_n)$. A propagator for the constraint $\sum_{i=1}^{n} B_i \geq \theta$, abbreviated by $\sum B$, is given in Algorithm 2. It first calculates the upper- and lower-bound of the summation function, and then checks if the constraint is satisfied or violated, and tries to propagate when possible. Depending on the operator $\lesseqgtr$ and on whether it is constrained by a variable or an actual value, a similar but different propagator can be implemented. More details on the implementation of such propagator variants can be found in a text book on constraint programming [12].

Another important concept that we will often use is the reification of constraints. A *reified constraint* $B \leftrightarrow \mathcal{C}$ is one that binds a boolean variable B to the *truth value* of the constraint $\mathcal{C}$. When it can be detected that the constraint is satisfied, or violated, this is propagated by setting $B = 1$, or $B = 0$ respectively. Likewise, if B is assigned to 1 or 0, constraint $\mathcal{C}$, respectively $\neg\, \mathcal{C}$, will be enforced.

In many of our models, we use reified boolean variables that are only used within an arithmetic constraint. We will use an *Iverson bracket* notation to shorten the notation of such constraints. The Iverson brackets $[\cdot]$ denote an operator which returns 1 if the constraint within brackets is satisfied, or 0 otherwise. Consider two vectors $V = V_1 \ldots V_n$ and $W = W_1 \ldots W_n$ of equal length n, and we wish to sum over the truth values B_x where $\forall x \in \{1 \ldots n\} : B_x \leftrightarrow V_x + W_x \geq 1$. We can abbreviate this sum to $\sum_x [V_x + W_x \geq 1]$.

Example 2 (CSP Continued): Let us illustrate how the search and propagation are performed for Example 1. We will abbreviate the domain value $\{0\}$ to 0, $\{1\}$ to 1 and $\{0, 1\}$ to ?, such that we can write the initial domains of the variables $\langle G, F, M, Ch_1, Ch_2\rangle$ as $\langle ?, ?, ?, ?, ?\rangle$. This is the initial domain and is depicted in the root of the search tree in Figure 1.

Initially none of the constraints can be propagated, so the search will pick a variable and assign a value to it. Which variable and value to pick can be defined by so-called variable and value order heuristics. The choice of the heuristics can have a huge impact on runtime efficiency, as different choices will lead to differently shaped search trees. We will use a general variable ordering heuristic that is known to perform good, namely to dynamically choose the variable that occurs in the most constraints. As value ordering we will first consider exclusion ($V = 0$) followed by inclusion ($V = 1$). In our example, initially the variables F and M both appear in 3 constraints, so the first variable of these two would be chosen and set to $F = 0$. This leads to the search node with domain values $\langle ?, 0, ?, ?, ?\rangle$, as shown in the upper left of Figure 1.

Because of $F = 0$, constraint $F + M \geq 1$ propagates that the mother has to join on the holiday trip ($M = 1$). In constraint $F \leftrightarrow M + G = 1$ the reified variable F is false, so the inverse constraint is posted: $M + G \neq 1$. Because of $M = 1$, this constraint propagates $M + G \neq 1 \Rightarrow 1 + G \neq 1 \Rightarrow G \neq 0$, so $G = 1$. At this point, the domain values are $\langle 1, 0, 1, ?, ?\rangle$. Constraint $G + F + M + Ch_1 + Ch_2 = 3$ can now be simplified: $1 + 0 + 1 + Ch_1 + Ch_2 = 3 \Rightarrow Ch_1 + Ch_2 = 1$. This constraint can not be simplified any further, so our partial solution remains $\langle 1, 0, 1, ?, ?\rangle$. The search now branches over $Ch_1 = 0$, which leads constraint $Ch_1 + Ch_2 = 1$ to propagate that $Ch_2 = 1$. This results in the first solution: $\langle 1, 0, 1, 0, 1\rangle$. The search backtracks to $\langle 1, 0, 1, ?, ?\rangle$ and branches over $Ch_1 = 1$, leading to the solution $\langle 1, 0, 1, 1, 0\rangle$. The search now backtracks to $\langle ?, ?, ?, ?, ?\rangle$ and branches over $F = 1$. Constraint $F + M \geq 1$ is now satisfied, so it is removed from consideration. In constraint $F \leftrightarrow M + G = 1$ the

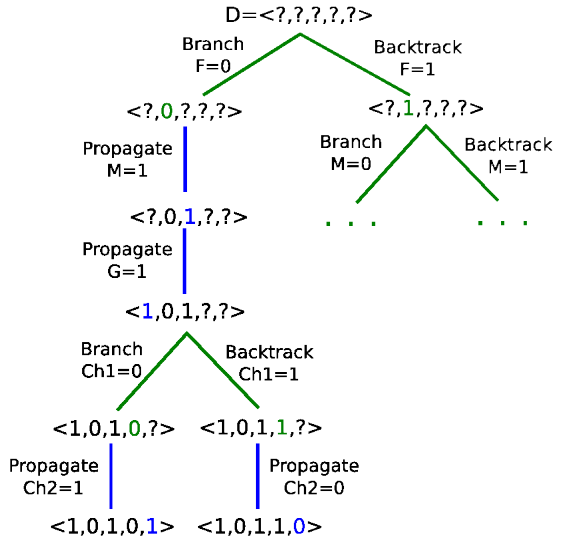

Fig. 1: Search tree for the holiday example.

reified variable F is true, so the reified constraint is replaced by $M+G=1$. This does not lead to any further propagation. Constraint $G+F+M+Ch_1+Ch_2=3$ is simplified to $G+M+Ch_1+Ch_2=2$, but remains bound-consistent. Since no more propagation can happen, the search procedures chooses the most constrained variable M and sets $M=0$: $\langle ?,1,0,?,?\rangle$. The CP solver will continue to alternate propagation and search in this way, until all solutions are found.

4 $k-$Pattern Set Mining using Constraint Programming

With its declarative modelling language and constraint-based search, constraint programming contains all the necessary elements to address the general problem of k-pattern set mining. Two questions remain; how can k-pattern set mining problems be specified in a CP system, and how effective will the search be?

In local pattern mining, the search space of itemset mining problems has size $O(2^n)$, where $n=|\mathcal{I}|$, the number of items in the database. Clearly, a naïve algorithm that would simply enumerate and test each pattern would not perform well. Thanks to the effective propagation of the constraints involved, for example the well-known frequency constraint, fast and efficient algorithms exist that do exhaustive search without having to enumerate each pattern. For k-pattern set mining, the search space has size $O(2^{nk})$ with n the number of items in the database and k the number of patterns in the set. Hence, to do exhaustive search, it becomes even more crucial to have constraints that effectively prune the search space.

In this section, we first study how the individual constraints of section 2.1 can be modelled in a constraint programming framework, and how effective their propagation will be. We then take a closer look at how the problem instantiations are modelled in the framework.

4.1 Modelling Constraints

Constraints are typically divided into two broad categories: local constraints and global constraints. A constraint is local when it is defined on one individual pattern, and global when it is defined on multiple patterns. This definition of a global constraint differs from the usual definition in constraint programming, where a global constraint indicates a constraint that relates a number of variables in a non-trivial way. During the study of the effectiveness of each constraint, we identify two new and special categories. Within the category of local constraints we identify local look-ahead constraints, and within the category of global constraints, the pairwise global constraints. This further distinction will give us a better understanding of the effectiveness of the constraints, which will help us to better understand the search behaviour of the models at large.

In our study all patterns are itemsets. We assume the itemset database $\mathcal{D}$ is represented as a binary matrix of size $|\mathcal{I}| * |\mathcal{T}|$ with $\mathcal{D}_{ti} = 1 \leftrightarrow (t,i) \in \mathcal{D}$. As in [7] we will represent a pattern's tuple $\pi = (I,T)$ by introducing a boolean variable for every item i and every transaction identifier t; in this way, an itemset I can be seen as a sequence of boolean variables I_i and a transaction set as a sequence of variables T_t. For instance, the pattern $(\{1,3\},\{1,2,5\})$, which has items 1 and 3, and is covered by transactions $1,2$ and 5 is represented as: $(\langle 1,0,1\rangle,\langle 1,1,0,0,1\rangle)$. A pattern set of size k simply consists of k such patterns: $\forall_{p=1..k} : \pi^p = (I^p,T^p)$.

4.1.1 Individual Pattern Constraints

Individual pattern constraints are by definition local constraints. We will not distinguish local constraints by being monotonic or anti-monotonic with respect to set inclusion, as common in constraint-based mining. In the constraint programming framework we always calculate bounds on the domains of variables, which is not tied to set inclusion/exclusion specifically. Note that from a constraint programming perspective, every constraint is monotonic with respect to the domain of the variables, as a propagator can only remove values from it.

We will review the constraints explained in Section 2.1.1 and elaborate on how they can be formulated in the constraint programming framework. We will also discuss their propagation power and categorise them as regular local or local look-ahead constraints. An overview can be found in Table 1.

Frequency constraint. The frequency constraint is the key constraint of frequent itemset mining. It is defined as the number of transactions that cover the itemset. In our CP formulation we have boolean variables T_t that represent whether the transaction with id t covers the itemset or not. The frequency of the itemset $\pi = (I,T)$ can then be computed as:

$$freq(\pi) = \sum_{t \in \mathcal{T}} T_t. \tag{32}$$

constraint category constraint name	set notation	CP
local look-ahead		
$coverage(\pi)$	$T = \varphi(I)$	$\forall t \in \mathcal{T} : T_t \leftrightarrow \sum_{i \in \mathcal{I}} I_i(1 - \mathcal{D}_{ti}) = 0$
$disjcoverage(\pi)$	$T = \alpha(I)$	$\forall t \in \mathcal{T} : T_t \leftrightarrow \sum_{i \in \mathcal{I}} I_i \mathcal{D}_{ti} > 0$
$closed(\pi)$	$I = \psi(T)$	$\forall i \in \mathcal{I} : I_i \leftrightarrow \sum_{t \in \mathcal{T}} T_t(1 - \mathcal{D}_{ti}) = 0$
$freq(\pi) \geq \theta$	$\lvert T\rvert \geq \theta$	$\forall i \in \mathcal{I} : I_i \rightarrow \sum_{t \in \mathcal{T}} T_t \mathcal{D}_{ti} \geq \theta$
$size(\pi) \geq \theta$	$\lvert I\rvert \geq \theta$	$\forall t \in \mathcal{T} : T_t \rightarrow \sum_{i \in \mathcal{I}} I_i \mathcal{D}_{ti} \geq \theta$
regular local		
$freq(\pi) \leq \theta$	$\lvert T\rvert \leq \theta$	$\sum_{t \in \mathcal{T}} T_t \leq \theta$
$size(\pi) \leq \theta$	$\lvert I\rvert \leq \theta$	$\sum_{i \in \mathcal{I}} I_i \leq \theta$

TABLE 1: Individual pattern constraints

constraint category constraint name	set notation	CP
pairwise		
$overlap(\pi^1, \pi^2)$	$\lvert T^1 \cap T^2\rvert$	$\sum_{t \in \mathcal{T}} [T_t^1 + T_t^2 = 2]$
$distinct(\pi^1, \pi^2)$	$\lvert (T^1 \cup T^2) \setminus (T^1 \cap T^2)\rvert$	$\sum_{t \in \mathcal{T}} [T_t^1 + T_t^2 = 1]$
$jaccard(\pi^1, \pi^2)$	$\frac{\lvert T^1 \cap T^2\rvert}{\lvert T^1 \cup T^2\rvert}$	$\frac{\sum_{t \in \mathcal{T}} [T_t^1 + T_t^2 = 2]}{\sum_{t \in \mathcal{T}} [T_t^1 + T_t^2 \geq 1]}$
$\min_{a<b} dist(\pi^a, \pi^b) \geq \theta$	$\min_{a<b} dist(\pi^a, \pi^b) \geq \theta$	$\forall a < b : dist(\pi^a, \pi^b) \geq \theta$
global		
$\sum_{a<b} dist(\pi^a, \pi^b) \lesseqgtr \theta$	$\sum_{a<b} dist(\pi^a, \pi^b) \lesseqgtr \theta$	$\sum_{a<b} V_{ab} \lesseqgtr \theta,$ $V_{ab} = dist(\pi^a, \pi^b)$
$\min_{a<b} dist(\pi^a, \pi^b) \leq \theta$	$\min_{a<b} dist(\pi^a, \pi^b) \leq \theta$	$\min_{a<b} V_{ab} \leq \theta,$ $V_{ab} = dist(\pi^a, \pi^b)$

TABLE 2: Redundancy constraints and their combinations

To constrain the frequency, we could simply constrain this sum, for example given a minimal frequency threshold θ: $\sum_{t \in \mathcal{T}} T_t \geq \theta$. This would be a **regular local constraint**: it calculates the lower and upper bound of a function on decision variables, and makes sure that they respect the threshold. The propagator for such a constraint was given in Algorithm 2. However, constraining the frequency in this way does not take the link between individual items and transactions into account. If during search an item occurs in less than the required number of transactions, it is easy to show that this individual item can never be part of a frequent itemset. Every time that the search would branch over this item by including it in the itemset, all the transactions that do not cover this item would be removed. After this removal, the number of transactions remaining is lower than the supplied threshold, so the frequency constraint would fail repeatedly. We can avoid this problem by formulating the following constraint. If an item is in the current itemset ($I_i = 1$), then the number of transactions that cover this itemset ($\sum_{t \in \mathcal{T}} T_t \mathcal{D}_{ti}$) must be above the frequency threshold:

$$\forall i \in \mathcal{I} : I_i \rightarrow \sum_{t \in \mathcal{T}} T_t \mathcal{D}_{ti} \geq \theta \tag{33}$$

When this constraint is posted, the solver will check for each individual item whether it can be part of a frequent itemset, at every node in the search tree. We call this kind of constraint a **local look-ahead constraint**, because it looks ahead for an individual variable and determines which values can still contribute to a valid solution in the future. A local look-ahead constraint like in equation (33) will often lead to better propagation than its regular local counterpart in equation (32). In a CP system, local look-ahead constraints can typically be modelled as reified constraints. We introduced reified constraints in Section 3.

Coverage constraint. The coverage constraint can be formulated as a local look-ahead constraint on every transaction T_t: a transaction is in the transaction set ($T_t = 1$) if the items not in the transaction ($\forall i : \mathcal{D}_{ti} = 0$) are not in the itemset ($\forall i, \mathcal{D}_{ti} = 0 : I_i = 0$); this can equivalently be written as

$$\forall t \in \mathcal{T} : T_t \leftrightarrow \sum_{i \in \mathcal{I}} I_i(1 - \mathcal{D}_{ti}) = 0. \tag{34}$$

The double implication $\leftrightarrow$ here guarantees that if there is an item that is not in the transaction ($\sum_{i \in \mathcal{I}} I_i(1 - \mathcal{D}_{ti}) \neq 0$), then that transaction is not covered; hence $T_t = 0$.

Closedness constraint. This constraint is very similar to the *coverage* constraint, but is formulated on items instead of transactions (see Table 1).

Size constraints. The minimum size constraint can be formulated as a local look-ahead constraint in a similar way as the minimum frequency constraint. In this case, look-ahead is done on the transaction variables: if a transaction does not have the minimum required number of items, then it can never cover an itemset of the minimum size, so the transaction can be removed.

Formulating a maximum size constraint as a look-ahead constraint is less useful; even if a transaction contains more than the required number of items, we cannot remove it from consideration, as not all these items necessarily have to be included in an itemset.

4.1.2 Redundancy Constraints

In Table 2 we give an overview of the redundancy constraints and how to combine them, as explained in Section 2.1.2.

Distance measures. In CP the cardinality of a set can in principle be calculated by summing 0/1 variables representing the elements in the set. However, to deal with redundancy, we often need to calculate the cardinality of a set which is the result of comparing two other sets. Representing the intermediary set with additional variables would make our notation cumbersome. For instance, to calculate the overlap $overlap(\pi^1, \pi^2) = |T_1 \cap T_2|$, we would first need to calculate the set $T_1 \cap T_2$, and then sum the variables representing this set. As a short-hand

constraint category constraint name	set notation	CP
global		
$freq(\Pi) \lesseqgtr \theta$	$\lvert\bigcup_\pi T\rvert \lesseqgtr \theta$	$\sum_{t\in\mathcal{T}} B_t \lesseqgtr \theta$, where $B_t = \left[(\sum_{p\in\{1..k\}} T_t^p) \geq 1\right]$
$accuracy(\Pi) \lesseqgtr \theta$	$(freq^+(\Pi) - freq^-(\Pi)) \lesseqgtr \theta$	$\left(\sum_{t\in\mathcal{T}^+} B_t - \sum_{t\in\mathcal{T}^-} B_t\right) \lesseqgtr \theta$, where $B_t = \left[\left(\sum_{p\in\{1..k\}} T_t^p\right) \geq 1\right]$
$area(\Pi) \lesseqgtr \theta$	$\lvert\bigcup_\pi (I \times T)\rvert \lesseqgtr \theta$	$\sum_{i\in\mathcal{I},t\in\mathcal{T}} B_{it} \lesseqgtr \theta$, where $B_{it} = \left[\left(\sum_{p\in\{1..k\}} [I_i^p + T_t^p = 2]\right) \geq 1\right]$

TABLE 3: Coverage and discriminative constraints

notation we will therefore combine the set operation and the size calculation as follows:

$$|T^1 \cap T^2| = \sum_{t\in\mathcal{T}} [T_t^1 + T_t^2 = 2]. \tag{35}$$

Here we count the number of transactions for which both T_t^1 and T_t^2 are 1 by using the Iverson bracket notation. Redundancy constraints measure the distance between two patterns, so they are by nature pairwise constraints.

Combining measures. The goal is not to constrain one pair of patterns, but all the pairs in a pattern set. The way in which this aggregation is done defines the complexity of constraining the redundancy of the overall pattern set. Constraining all the distances at once would result in one large global constraint. An example of this is when we would sum over all the pairwise distances: $\sum_{a<b} dist(\pi^a, \pi^b) \lesseqgtr \theta$.

This constraint would not propagate very well as a change in one pattern will not directly influence the other patterns. Typically many of the distances would have to be known before the constraint can propagate a change on the other distances. A better solution would be to constrain every distance individually. Consider the case where we want to constrain the smallest pairwise distance to be larger than a certain threshold θ: $\left(\min_{a<b} dist(\pi^a, \pi^b)\right) \geq \theta$. In this case, we can put a constraint on each pair separately: if the smallest distance has to be larger than the threshold, then all distances have to be larger than the threshold: $\forall a < b : dist(\pi^a, \pi^b) \geq \theta$. In this case, we can decompose the global constraint in a number of independent pairwise constraints. This difference in propagation strength motivates us to discriminate pairwise constraints from regular global constraints.

4.1.3 Coverage and Discriminative Constraints

In Table 3 we list the coverage and discriminative constraints discussed in Section 2.1.3 and Section 2.1.4. The cover of the entire pattern set depends on the cover of every individual pattern $T^\Pi = T^1 \cup \cdots \cup T^k$. If a transaction is covered by one pattern, it is covered by the pattern set. Hence, a transaction t is covered iff $\exists \pi \in \Pi : T_t = 1$. In constraint programming we can model this by introducing a temporary variable B_t for every transaction t, where $B_t \leftrightarrow (\sum_{p\in\{1\ldots k\}} T_t^p) \geq 1$. Coverage and discriminative constraints, which we originally defined on the transaction sets of individual patterns, can also be defined on such temporary variables. Because of the indirect relation between patterns through these temporary variables, coverage and discriminative constraints are categorised as regular global constraints.

A special case is the *area* constraint, for which we need to calculate the number of ones in the matrix that is covered by the set of patterns. We can model this by introducing a temporary variable B_{it} for every element in the matrix. $B_{it} = 1$, iff at least one tile covers that element, that is, $\exists \pi \in \Pi : (I_i = 1 \wedge T_t = 1)$. Equivalently $B_{it} = \left[\left(\sum_{p\in\{1..k\}} [I_i^p + T_t^p = 2]\right) \geq 1\right]$. This constraint can not propagate the truth-value of B_{it} well: if $B_{it} = 1$ then it should propagate that at least 1 pattern covers this element, but this is only possible of all but one pattern are known not to cover it. Additionally, the area constraint contains $|\mathcal{I}| * |\mathcal{T}|$ such variables, where $|\mathcal{T}|$ is typically large. Hence, we can expect this constraint to have particularly weak propagation.

4.1.4 Symmetry Breaking Constraints

We model the set of k patterns in constraint programming by means of an array of k patterns. An artefact of this is that syntactic variations of the same pattern set can be generated, for example (π_1, π_2) and (π_2, π_1). This is something well-known in constraint programming that can be solved by means of so-called *symmetry breaking constraints* [12]. In many cases, symmetry breaking constraints impose a strict ordering on the patterns in the pattern set.

There are many ways to impose an ordering on the array of patterns. The most straightforward way is to impose a simple lexicographic ordering on the patterns in the array:

$$symm\_breaking(\Pi) : \pi^1 < \pi^2 < \ldots < \pi^k \tag{36}$$

Given that the order constraint can be enforced between every pair of patterns, this constraint falls in the category of *pairwise constraints*.

There are some design decisions to be made when ordering patterns. Of course, one can impose a lexicographic ordering on the itemsets of the patterns in the array. In case every pattern is a closed itemset, the itemset uniquely defines the transaction set and the transaction set uniquely defines the itemset. Hence

instead of lexicographically ordering the itemsets, one can also order the transaction sets. As the transaction set is typically larger than the itemset, this could lead to better propagation as more variables are involved.

Alternatively, one could order the patterns primarily on some property like frequency or accuracy, and use a lexicographic order on the item- or transaction sets of the patterns to break ties. Finally there is also the choice to post the order constraints only on subsequent patterns, or between all pairs of patterns. Posting them between all pairs requires $(n-1)(n-2)/2$ additional constraints, but could result in some additional pruning.

4.2 Modelling Instantiations

The constraints introduced in the previous section allow us to model all the mining problems that were introduced in Section 2.2. Essentially, to obtain a complete CP model, we need to enter the appropriate CP formulations of constraints in the problem descriptions of Section 2.2. Let us illustrate this for the problem of concept learning.

One step: When we fill in the formulas of the previous section in the model of equation (27), we obtain this CP model:

$$\begin{aligned}
&\forall p \in \{1,\ldots,k\}\\
&\qquad \forall t \in \mathcal{T} : T_t^p \leftrightarrow \sum_{i\in\mathcal{I}} I_i^p(1-\mathcal{D}_{ti}) = 0, \quad (Coverage)\\
&\qquad \forall i \in \mathcal{I} : I_i^p \leftrightarrow \sum_{t\in\mathcal{T}^+} T_t^p(1-\mathcal{D}_{ti}) = 0, \quad (Closed^+)\\
&\forall t \in \mathcal{T} : B_t = [(\sum_{p\in\{1..k\}} T_t^p) \geq 1],\\
&\qquad \sum_{t\in\mathcal{T}^+} B_t - \sum_{t\in\mathcal{T}^-} B_t \geq \theta, \quad (Accurate)\\
&\qquad T^1 < T^2 < \ldots < T^k \quad (Symm.Br.)
\end{aligned}$$

This model captures a problem that involves k patterns at the same time; constraint programming systems provide a strategy for finding optimal solutions to this problem. The important advantage of this model is that it allows a one step solution to the concept learning problem.

Two step: Nevertheless, one can also use CP systems to find an optimal solution in two steps. Closed-on-the-positive itemsets are also sufficient in this case. First, all itemsets that fulfil the $coverage(\pi)$ and $closed^+(\pi)$ constraints are mined. Using all itemsets found, a new transactional database is created in which every item represents an itemset. On this new transactional database another pattern mining problem is solved, with the following constraints:

$$\begin{aligned}
&disjcoverage(\pi),\\
&size(\pi) = k,\\
&accuracy\left(freq^+(\pi), freq^-(\pi)\right) \geq \theta.
\end{aligned}$$

Each resulting itemset corresponds to a set of local patterns and is hence a pattern set. The constraints enforce the desired size k and accuracy θ. This approach is similar to the approach chosen in [13], where additionally a frequency constraint is used. Our hope is that the single-step approach outperforms this more traditional step-wise approach. Whether this is the case will be explored in the next section.

	Transactions	Items	Density	Class distr.
anneal	812	53	42%	77%
audiology	216	148	45%	26%
hepatitis	137	44	50%	81%
lymph	148	60	38%	55%
primary-tumor	336	31	48%	24%
soybean	630	50	32%	15%
tic-tac-toe	958	27	33%	65%
vote	435	48	33%	61%
zoo	101	36	44%	40%

TABLE 4: Dataset properties

5 Experiments

As shown in the previous section, constraint programming offers a powerful framework that fits the general problem of k-pattern set mining. Constraint programming has a generic search strategy of propagation and exhaustive search. The big question that we will try to answer in this section is: *How suitable is a generic CP approach, which uses exhaustive search, for pattern set mining?*

We have argued for the importance of constraints that propagate well, as this is needed to keep the search space manageable. In this section, we perform an experimental study on each of the introduced pattern set mining tasks. For each task, we will study the performance of the CP approach, and link these results to the constraints involved in the model.

The aim of these results is to gather a better understanding in the suitability of CP systems for a broad range of tasks. The goal is not to evaluate individual tasks. Constraint programming offers us a single framework for all k-pattern set mining problems. Although each of the tasks can be modelled in the CP framework, that does not necessarily mean it can be solved efficiently. Hence, our goal is to study the opportunities and limitations of the constraint programming framework for k-pattern set mining in general.

We focus our study around the following questions:

1) how does the proposed one step approach compare to the two step procedure?
2) how does the k-pattern set mining approach scale to the different tasks?
3) what is the relation between the performance of a model and the constraints involved?

We study these questions by performing experiments for each of the four tasks of interest, that is, concept-learning, clustering, tiling and rediscription mining. We conclude with some general conclusions.

For the experiments we used Gecode [14], an open and efficient constraint programming solver. It includes propagators for all the constraints used in this paper.

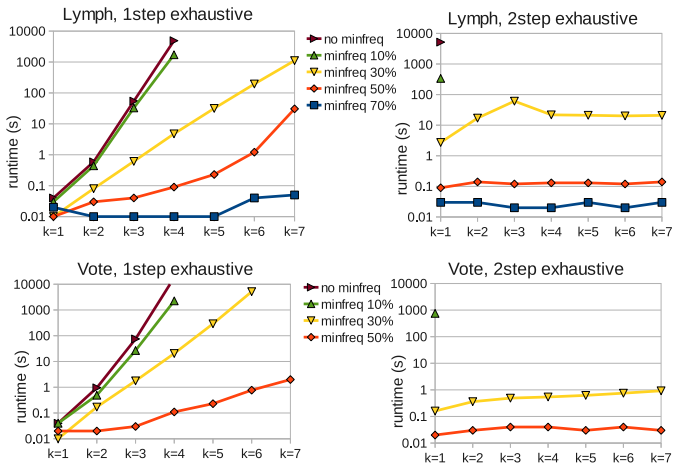

Fig. 2: Comparing a one step exhaustive search with a two step exhaustive search to concept learning.

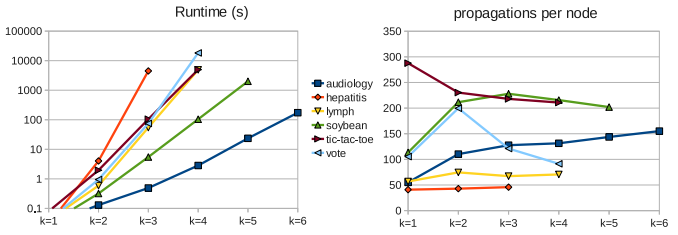

Fig. 3: Runtime and number of propagations per node for the concept learning setting.

For constraints of the form $\sum[B^1 + B^2 \lesseqgtr \alpha]$ we added a simple propagator that directly calculates the sum at large, thus avoiding to store an auxiliary variable for every single reified sum. The datasets used are from the UCI Machine Learning repository [15] and were discretised using binary splits into eight equal-frequency bins. The majority class was chosen as the positive class. The properties of the resulting datasets are listed in Table 4. Experiments were performed on PCs running Ubuntu 8.04 with Intel(R) Core(TM)2 Quad CPU Q9550 processors and 4GB of RAM. All datasets and source code used will be available online at `http://dtai.cs.kuleuven.be/CP4IM/` upon publication of this article, as are our other results on combining pattern mining and constraint programming.

5.1 $k-$term DNF Learning and Concept Learning

We focus on the concept learning setting in which we want to maximise the accuracy of the pattern set. In case a k-pattern set is found that covers all positive transactions and none of the negative ones, this setting is identical to pure k-term DNF learning.

We start by investigating the first question: how does the proposed one step approach compare to the two step procedure (Q1)? We compare both in our CP framework. In the two step approach, detailed in Section 4.2, first all patterns given a minimum frequency threshold are mined, and in the second step the best combination of k patterns is sought using constraint programming. In the one step approach we search for the k pattern set directly; a minimum frequency constraint is not needed, but we add it for comparison. Figure 2 shows the result of this experiment for the lymph and vote datasets. The two step approach performs very good for high frequency thresholds such as 50% and 70%; for low thresholds the two step approach can not handle the amount of candidate patterns that is generated in the first step. In our experiments, it could only handle up to a few thousand generated patterns. The one step approach on the other hand does not rely on the frequency constraint; even when there are potentially large amounts of local patterns, and hence the two-step method fails, the one step method can use the global constraints, up to certain values of k to reduce the size of the search space.

Next, we investigate the question how the approach scales to different tasks (Q2). In Figure 3 we study the k-pattern set mining problem of concept learning without a minimum frequency threshold on different datasets. For $k = 1$ we are able to find the single concept that best describes the data in less than a second. For $k = 2$ the two best concepts are also found in a reasonable amount of time, but for larger k the run times quickly become very large and the scalability is limited.

This can be explained by looking at the types of constraints involved, as mentioned in our third question (Q3). Overall, the concept learning model consists of the coverage and closed local look-ahead constraints which we already found to perform well in previous work [8], as well as the global accuracy constraint and pairwise lexicographic ordering constraints (no frequency constraint are needed). The case $k = 1$ is special, as there are no ordering constraints and the accuracy constraint is local. For $k = 2$, there is a pairwise ordering constraint; the accuracy constraint is also pairwise as it is expressed on only two patterns. Starting from $k = 3$ more pairwise constraints are added, and the accuracy constraint becomes a regular global constraint.

Our hypothesis is that the more patterns are included in the pattern set, the less efficient the propagation of the global constraint becomes and hence the longer the search will take. To test this hypothesis, we plot the average number of constraint propagations per node of the search tree on the right of Figure 3. We observe that the number of propagations per node decreases or stays the same, except for the audiology dataset for which it increases moderately. Knowing that the number of nodes in the search tree grows quickly for increasing k, a lack of increase in propagation means that many of those nodes will have to be visited. Hence for large k, the global accuracy constraint and the pairwise ordering do not allow for sufficient propagation to make the search feasible.

5.2 Conceptual clustering

In conceptual clustering, one is interested in finding clusters that are described by conceptual descriptions.

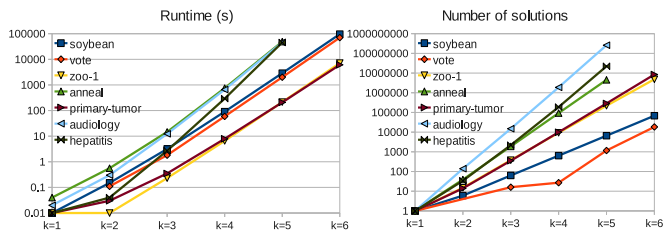

Fig. 4: Runtime and number of conceptual clusterings for varying k.

In our case the conceptual description is an itemset, and the corresponding transactions constitute the cluster. The goal is to find a clustering with a certain number k of non-overlapping clusters.

To address the question how CP scales to different tasks (Q2), we first consider the differences between the constraint satisfaction setting (in which we wish to find all solutions that satisfy the constraints), and optimisation settings, as given in Section 2.2.2.

Figure 4 shows the run time and the number of non-overlapping clusterings found in different datasets, for varying k. We observe that many non-overlapping clusters exist. This is explained by the high dimensionality of our binary data. For increasing k, the runtime and the number of clusterings found increases exponentially. A similar phenomenon occurs in traditional itemset mining: weak constraints, in our case a high value of k, lead to a combinatorial explosion where most time is spent on enumerating the solutions.

When looking at the resulting clusterings in more detail, we noticed that many of the clusterings include patterns that cover only one transaction. For $k = 3$, it is common to have one large cluster covering most of the examples, one medium sized cluster, and one cluster that covers only one transaction. Such clusterings can be considered less interesting than those in which the clusters cover about the same number of transactions. To avoid this, we introduced the optimisation settings in which we search for more balanced clusterings.

The left figures in Figure 5 illustrate the first optimisation setting, in which we maximise the minimum cluster size, while the right figures illustrates the second setting, in which we minimise the cluster range. In both cases, the figures show the runtime, minimum cluster size and cluster size range for different sizes k.

We see that the size of the smallest cluster decreases as the number of clusters k increases. The range of the clusters differs depending on the specific dataset. However, there is a decreasing trend in the range, indicating that a larger number of clusters can more evenly cluster the data.

To assess the influence of the types of constraints (Q3), it is also useful to compare the left and right figures in Figure 5. We see that the second approach, in which the range is optimised, scales less well for increasing k, although the solutions found are almost always the same. The difference can be explained by the

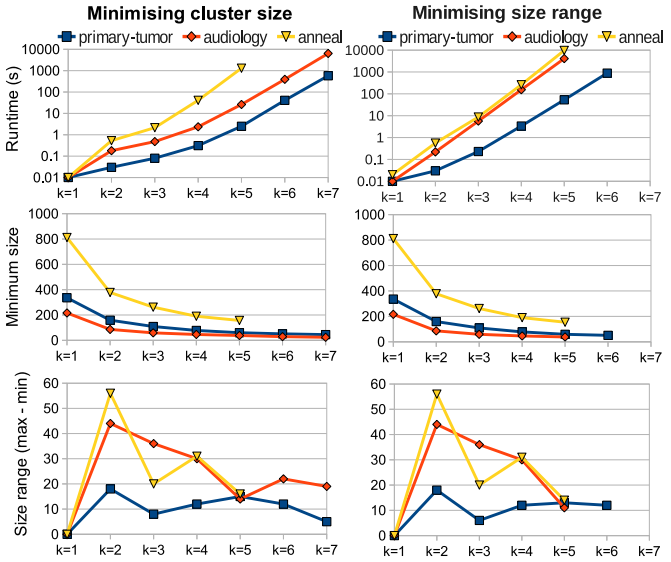

Fig. 5: Mining a pattern set clustering with k conceptual clusters. Left: when optimising the minimum cluster size. Right: when optimising the cluster size range.

difference between the 'minimum size' constraint and the 'size range' constraint. The minimum size constraint acts as a local frequency constraint once one candidate solution has been found. In all later solutions, each cluster has to contain at least as many examples as the smallest cluster of the earlier solution. The size range constraint, on the other hand, is a global constraint that operates on the minimum and maximum value over all clusters in the clustering, and does not reduce to a simple local frequency constraint during the search. We can conclude that if there is a choice between local and global constraints, then local constraints should be preferred as they propagate more effectively.

5.3 $k-$Tiling

In section 4.1.3 we presented a formulation of the area constraint and discussed why this constraint will propagate badly. Not only does it depend on $|\mathcal{I}|*|\mathcal{T}|$ variables, but each of these variables has only a weak relation to the patterns in the set. For this reason, we do not expect the $k-$tiling model to perform well, making it a good setting to investigate the limits of our CP approach (Q2).

We will compare finding the optimal k-tiling to finding a greedy approximation of it. The greedy method iteratively searches for the single pattern (tile) that covers the largest uncovered area in the database, where we implemented the search for the best scoring tile also in the CP system; this iterative greedy algorithm is known to be approximate the optimum [6]. The iterative procedure continues until k patterns are found.

Table 5 shows the maximum k for which an optimal k-tiling was found within a time limit of 6 hours per run, and the corresponding area. We also report the area found by a greedy k-tiling algorithm for the same value of k. Note that the highly efficient greedy algorithm

	pattset k-tiling		greedy k-tiling
	max k	area for this k	area for this k
soybean	3	4447	4357
zoo	3	772	749
lymph	2	1445	1424
primary-tumor	2	1841	1841
tic-tac-toe	2	876	876
vote	2	1758	1758

TABLE 5: Maximum k and area for the k-tiling problem on multiple datasets, with a timeout of 6 hours. The right column shows the area for the same k, when using a greedy tiling algorithm.

always finished within 10 seconds. Only for very small values of k could an optimal k-tiling be found. Although the CP method finds the optimal solution, the greedy method's area was always close to or equal to it. In the global CP approach, the area constraint is too weak to prune the search space significantly, and the search space is too big to enumerate exhaustively. This shows that for k-tiling, an exhaustive search method is not advised, at least not using the area constraint of Section 4.1.3.

5.4 Redescription mining

We consider the case where the data consists of two partitions; they both range over the same set of transactions but consist of two different sets of items. A description is a pattern defined on one of the partitions of items. We are interested in finding redescriptions, namely two patterns from the different partitions that cover many or all of the same examples. In our experimental setting, we randomly partitioned the attributes in two classes.

To investigate (Q1) we compare two approaches for finding the most frequent exact redescription. The first approach is a one step approach and models the entire problem in CP. In the solution, the patterns cover exactly the same set of examples and this set is the largest of all redescriptions. In the second approach, we first find all closed itemsets that have at least one item in each partition. Redescriptions can be found by postprocessing these itemsets, as the union of two closed itemsets with equal coverage must be a closed itemset in the original data as well. Table 6 shows the run time needed for the one step approach (column 2) and for the first step of the two step approach (column 4). Finding all closed itemsets already takes more time than finding the global optimal solution in one step for most datasets. In the two step approach, each of the patterns would also have to be post-processed. This would increase the difference in runtime even further, especially for datasets with many solutions. Hence, on this problem the one step approach is again more promising.

To investigate (Q2) we compare the several settings for redescription mining introduced in Section 2.2.4. The first three columns of Table 6 list the result for the most frequent exact redescription. Figure 6 shows results from searching for the best redescription under different frequency thresholds, where we use the *distinct* measure explained in Section 4 as quality measure. A

	exact redescriptions		closed sets	
	time (s)	rel. freq.	time (s)	nr. solutions
hepatitis	0.08	4.38%	14.12	1824950
primary-tumor	0.02	0.60%	0.34	29395
vote	0.1	0.92%	0.4	32669
soybean	0.13	3.81%	0.11	2769
zoo	0.02	39.60%	0.03	3029

TABLE 6: Run times for redescription mining settings; on the left, runtime and relative frequency when searching for the exact redescription covering most examples; on the right, runtime and number of patterns found when mining all closed sets forming a redescription.

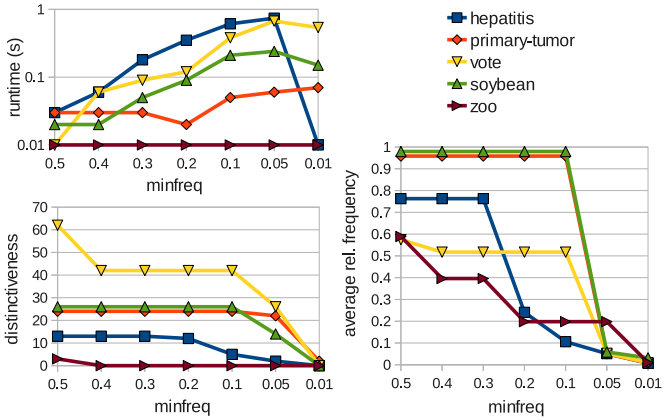

Fig. 6: Run time, distinctiveness and average frequency of patterns when searching the least distinct redescription given a minimum frequency threshold.

distinctiveness of zero corresponds to an exact redescription. The run times for finding the redescriptions are generally low in these settings. Except for the zoo dataset, the relative frequency of the redescriptions in Table 6 is low (column 3). For these datasets, there are no exact redescriptions covering many transactions. Figure 6 shows the result for different minimum frequency thresholds. Low minimum frequency thresholds lead to more similar patterns but possibly less interesting ones. Higher thresholds lead to more distinct patterns which are usually a lot more prominent in the data. When we study the run times of these results, we can draw the following conclusions with respect to (Q3). The low run times can be attributed to the constraints at hand: the usual coverage and closedness local look-ahead constraints, a pairwise constraint between the two transaction sets, and a minimum frequency local look-ahead constraint for the branch-and-bound search in the first setting. The local look-ahead constraints are known to propagate well, and the pairwise constraint is able to immediately propagate changes in one transaction set to the other. Posting these constraints leads to very effective propagation and thus fast execution times. In the second setting, like the constraint for exact redescriptions, the *distinct* constraint is also a pairwise constraint. Although less effective, the fact that it is pairwise allows the constraint to effectively propagate changes between two transaction sets too.

5.5 Discussion

We have focussed our study of the suitability of exhaustive search for pattern set mining on three questions.

The first question was how the proposed one step approach compared to a more traditional two step procedure. The results on the concept learning and redescription mining problems showed that a one step exhaustive search is more efficient than a two step exhaustive search. In the two step approach the bottleneck is the large number of patterns found in the first step, in which the second step then has to search for a subset. The one step approach, on the other hand, can use the constraints of the global model to reduce the number of patterns that need to be considered. Compared to greedy strategies, however, the approach is much slower and rarely finds significantly better results.

The second question was how the k-pattern set mining approach scales to different tasks. There are significant differences in the scalability of the approach, depending on the task at hand. Even for the same task, as we saw for conceptual clustering, alternative formulations that find the same solutions can have different runtime behaviour.

The third question was what the relation between the performance of a model and its constraints was. The differences in runtime depend on the constraints used. The k-tiling problem was the task with the fewest constraints, involving the largest amount of variables. Consequently, it scaled very badly and only found solutions for the lowest values of k. Concept learning, with its better propagating *accuracy* constraint, performed better than k-tiling, although its scalability was also limited. Conceptual clustering performed better, especially when formulating it such that the optimisation constraints are local constraints. Lastly, the best results were obtained for the redescription mining task, which contained only local look-ahead and pairwise constraints.

A key question in this section was whether a generic CP approach that uses exhaustive search is also suitable for pattern set mining. We found that the answer depends on the constraints involved in the model of the mining task. As is the case in traditional pattern mining, when there are only local constraints, most of which are local-lookahead constraints, then the generic CP approach is possible. Interestingly, when there are local lookahead constraints, as well as pairwise constraints, the approach is feasible too. This was the case for the redescription mining and our models of conceptual clustering. In fact, when $k = 2$ all tasks contain only pairwise constraints. Indeed for all tasks for $k = 2$, solutions were found in an acceptable time.

In the presence of non-pairwise global constraints, the use of the proposed framework is limited to small values of k. It appears that a crucial factor is whether constraints are local look-ahead constraints or pairwise constraints. Nonetheless, we believe that our study can lead to a better understanding of constraints, which can help the development of heuristic approaches as well. For example, the greedy approach that iteratively called the CP solver to mine for local patterns performed very well. This raises the question whether a general approach using such a large neighbourhood search [16] strategy could be devised for the k-pattern set mining problem at large.

6 Related Work

There are two distinctive features in our approach. First, the framework for k-pattern set mining can – in contrast to most other approaches in data mining and machine learning – be used to tackle a wide variety of tasks such as classification, clustering, redescription mining and tiling. Second, our work sets itself apart by using a one step exhaustive approach, while other techniques to mining pattern sets typically use a two step approach or a heuristic one step approach.

Similar to our work is the recent work of Khiari et al. [17] who propose to mine for n-ary patterns, patterns containing n patterns, using constraint programming. Our independently developed work goes well beyond theirs by covering a much wider range of tasks and by providing a profound study on the propagation power of the constraints. In retrospect, the good efficiency results they achieved can be explained by our results: they performed experiments with either 2 or 4 patterns, in which all constraints were local-lookahead or pairwise constraints. We have observed and explained why this is essential for an exhaustive CP approach to be efficient.

We will first review local techniques to reducing redundancy in the collection of mined patterns. We then in turn discuss global exhaustive two step techniques and global heuristic techniques.

6.1 Local Techniques

Techniques for removing redundancy at the local level typically focus on finding patterns that form a condensed representation ([18], [19], [20], [21]). Condensed representations range from exact representations for which the frequency of each pattern can be reconstructed, to lossy representations that discard information regarding frequency [20]. Whether a pattern is part of a condensed representation depends on its sub- and supersets. This can often be efficiently determined during search, making many condensed representations an adequate tool for decreasing not only the number of patterns found but also the computational resources needed. However, there are no guarantees as to the size of the reduction, so condensed representations solve the redundancy problem only partly. Because of its advantages, condensed representations are usually mined for in the first step of typical two step algorithms. Our approach considers only closed frequent patterns. It is possible to use other condensed representations such as the ones defined in [8], although it could be that the globally optimal pattern set does not exist for some condensed representations.

6.2 Global Exhaustive Two step Techniques

The framework for k-pattern set mining that we introduced builds upon the notion of exhaustive pattern set mining by De Raedt and Zimmermann [13]. They provided a general definition of two step constraint-based pattern set mining by exploiting the analogies with local pattern mining. The key differences with the present approach is that their work assumes a two step procedure, that is, it actually is centred around the computation of

$$\text{Th}(\mathcal{L}, p, \mathcal{D}) = \{\Pi \subseteq \text{Th}(\mathcal{L}, p', \mathcal{D}) \mid p(\Pi, \mathcal{D}) \text{ is true}\}$$

in which first a local pattern mining step is performed, resulting in the set $\text{Th}(\mathcal{L}, p', \mathcal{D})$ and then subsets containing such patterns are searched for. A further difference with the present approach is that we look for sets containing a fixed number of patterns. While this is more restrictive it is – in our opinion – essential from a computational perspective. Lastly, because the approach of [13] is derived from local pattern mining, it suffers from the same problems as the original pattern mining algorithms, namely an overwhelming amount of pattern sets, many of which are redundant. To avoid this problem, we focussed on finding the optimal pattern set, according to some measure, directly. By removing this optimisation criterion, our approach can be used to find all pattern sets too.

Related to the interpretation of a pattern set as a DNF formula is also the BLOSOM framework [22], which can mine for all DNF and CNF expressions in a binary dataset. BLOSOM uses the notion of closed DNF and minimal DNF to minimise the logical redundancy in the expressions. However, a two step approach is again used in which first (variants of) frequent patterns are searched and later post-processed.

6.3 Global Heuristic Techniques

While [13] used an exhaustive two step approach to finding pattern sets, there are numerous heuristic approaches to finding global pattern sets that first perform a local pattern mining step and then heuristically post-process the result ([1], [23]) (see [24] for an overview). Thus the second step does not guarantee that the optimal solutions are found. Furthermore, these approaches usually focus on one specific problem setting such as classification. For instance, CBA [1] first computes all frequent itemsets (with their most frequent class label) and then induces an ordered rule-list classifier by removing redundant itemsets. Several alternative techniques (for instance, [23], [25]) define measures of redundancy and ways to select only a limited number of patterns. Among them, one step greedy methods are also common [26], [25]. Constructing a concise pattern set for use in classification can be seen as a form of feature selection.

Another related problem setting is that of finding a good compression of the entire collection of frequent patterns. There exist many different approaches such as clustering the collection of frequent patterns [27], finding the patterns that best approximate the entire collection [28], ordering the patterns such that each prefix is a good summary [29], ordering according to statistical p-value [30], and more. By nature, these techniques also work in two steps: first find all patterns given a minimum frequency threshold, then compress that collection of patterns.

Techniques also exist that try to compress the dataset rather than the collection of patterns. For example, the KRIMP algorithm [31] uses a Minimal Description Length based global measure of compression. The compressed set of patterns covers all transactions, as we also often required in this work. Alternatively, a greedy covering technique similar to the one used in [28] could be applied on the dataset. However, in both cases, again a heuristic algorithm is used and the size of the selected pattern set is unbounded.

In [32], Knobbe and Ho present the concept of a *pattern team* as a small subset of patterns that optimises a measure. They identify a number of intuitions about pattern sets, and four measures that satisfy them; two of these are unsupervised measures while the other two are supervised. The first supervised measure uses a classifier and cross validation to assess the quality of a pattern team, which is beyond the scope of our work. The second supervised measure is area under the ROC curve. We showed in [33] that it is possible to exhaustively find the set of all patterns on the ROC convex hull, even without restricting the set to a size k. The work in [32] differs from ours as they mostly focus on how quality measures cover certain intuitions while we focus on how to express many constraints in one framework and also on the impact of these constraints on exhaustive search.

Lastly, declarative languages for data mining tasks have been proposed before [34], [35], [36], usually modelled after the idea of inductive databases [37]. Rather than being a query language, we propose a modelling language closer to mathematical programming, in which constraints can be decomposed, alternative formulations and optimisation criteria can be expressed.

7 Conclusions

We introduced the k-pattern set mining problem. It tackles pattern mining directly at a global level rather than at a local one. Furthermore, it allows for the specification of many different mining tasks in a uniform way by varying the involved constraints.

We used constraint programming methodologies and solvers for addressing the k-pattern set mining problem in a general way. This was realized by mapping each of the presented k-pattern set mining tasks to models in the CP framework. However, often multiple equivalent modelling strategies are possible and the model that is chosen can have a large influence on the performance of the solver. To provide guidelines for choosing amongst alternative formulations, we categorized the individual

constraints on the basis of their propagation power. Except for the natural distinction between local constraints on individual patterns and global constraints on multiple patterns, we identified two special categories: local-lookahead constraints, a type of local constraint, and pairwise constraints, which are global constraints restricted to two patterns. Constraints in these two categories propagate better than regular local or global constraints. Propagation strength is important for techniques that rely on exhaustive search, such as CP.

Using the CP framework, we evaluated the feasibility of exhaustive search for pattern set mining. We found that a one-step search method is often more effective than a two-step method in which an exhaustive search is performed in both steps. In general, we can conclude that the feasibility of exhaustive search depends on the constraints involved. When the problem specification consists mostly of local-lookahead and pairwise constraints, then exhaustive search can be capable of finding the optimal solution. However, in case there are other, non-pairwise global constraints, an exhaustive solution method will not perform well.

Nonetheless, we believe that the study of individual constraints and of how constraints interact can lead to a better understanding of the relationship between local and global constraints. Further studies, for example on what other global constraints can be decomposed into pairwise constraints, can lead to advances of exhaustive as well as heuristic algorithms. In general, the study of the relation between local and global constraints, not necessarily in a one step exhaustive framework, merits further study. We hope to have contributed to this by providing a framework for rapid prototyping of existing and new problems, as well as for the controlled study of individual constraints.

There are several interesting issues for future work. First, we have presented a general problem formulation of pattern set mining and studied a number of constraints and instantiations. There is a whole range of other pattern set mining tasks and constraints that can also be studied within the k-pattern set mining framework. The usefulness of constraints, especially when using exhaustive search, depends on how well constraint can propagate. Further study on the propagation power of constraints is required. Interesting questions include: what global constraints can be decomposed into pairwise constraints? Are better propagation algorithms possible for existing (global) constraints?

We have investigated the opportunities and limitations of using constraint programming to solve the general k-pattern set mining problem. Constraint programming uses exhaustive search and the feasibility of the CP approach depended on the constraints involved. It is an open question whether other solvers can be used to solve the general k-pattern set mining problem given only its description in terms of constraints. In the artificial intelligence community, a number of constraint-based heuristic approaches have been developed that follow a similar declarative approach as constraint programming. Prominent examples include large neighbourhood search [16] and constraint-based local search [38]. The authors believe such methods could provide a general solution for tasks where the CP approach currently lacks.

The presented framework implements a part of our long term vision to develop generally applicable declarative data mining tools. Possibilities for future work include the extension towards other data mining tasks that are not necessarily connected to pattern mining, and linking the framework to the many data mining methods based on mathematical programming. In the long term, this should allow data mining experts to solve data mining problems by specifying a series of models in an declarative data mining language with an integrated solver environment [39].

Acknowledgements We would like to thank the anonymous reviewers for their in-depth feedback. This work was supported by a Postdoc and a project grant from the Research Foundation—Flanders, project "Principles of Patternset Mining" as well as a grant from the Institute for the Promotion and Innovation through Science and Technology in Flanders (IWT-Vlaanderen).

REFERENCES

[1] B. Liu, W. Hsu, and Y. Ma, "Integrating classification and association rule mining," in *KDD*. AAAI Press, 1998, pp. 80–86.
[2] W. Li, J. Han, and J. Pei, "CMAR: Accurate and efficient classification based on multiple class-association rules," in *ICDM*. IEEE Computer Society, 2001, pp. 369–376.
[3] M. J. Kearns and U. V. Vazirani, *An introduction to computational learning theory*. Cambridge, MA, USA: MIT Press, 1994.
[4] D. H. Fisher, "Knowledge acquisition via incremental conceptual clustering," *Machine Learning*, vol. 2, no. 2, pp. 139–172, 1987.
[5] L. Parida and N. Ramakrishnan, "Redescription mining: Structure theory and algorithms," in *AAAI*. AAAI Press, 2005, pp. 837–844.
[6] F. Geerts, B. Goethals, and T. Mielikäinen, "Tiling databases," in *Discovery Science*. Springer, 2004, pp. 278–289.
[7] T. Guns, S. Nijssen, and L. D. Raedt, "Itemset mining: A constraint programming perspective," *Artificial Intelligence*, vol. 175, no. 12-13, pp. 1951 – 1983, 2011.
[8] L. De Raedt, T. Guns, and S. Nijssen, "Constraint programming for itemset mining," in *KDD*. ACM, 2008, pp. 204–212.
[9] J. Fürnkranz and P. A. Flach, "Roc 'n' rule learning-towards a better understanding of covering algorithms," *Machine Learning*, vol. 58, no. 1, pp. 39–77, 2005.
[10] S. Morishita and J. Sese, "Traversing itemset lattice with statistical metric pruning," in *PODS*. ACM, 2000, pp. 226–236.
[11] G. C. Garriga, P. Kralj, and N. Lavrac, "Closed sets for labeled data," *Journal of Machine Learning Research*, vol. 9, pp. 559–580, 2008.
[12] F. Rossi, P. van Beek, and T. Walsh, *Handbook of Constraint Programming*. Elsevier, 2006.
[13] L. De Raedt and A. Zimmermann, "Constraint-based pattern set mining," in *SDM*. SIAM, 2007, pp. 1–12.
[14] Gecode Team, "Gecode: Generic constraint development environment," 2010. [Online]. Available: http://www.gecode.org
[15] A. Frank and A. Asuncion, "UCI machine learning repository," 2010. [Online]. Available: http://archive.ics.uci.edu/ml
[16] P. Shaw, B. De Backer, and V. Furnon, "Improved local search for cp toolkits," *Annals of Operations Research*, vol. 115, pp. 31–50, 2002.
[17] M. Khiari, P. Boizumault, and B. Crémilleux, "Constraint programming for mining n-ary patterns," in *CP*. Springer, 2010, pp. 552–567.
[18] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal, "Discovering frequent closed itemsets for association rules," in *ICDT*. Springer-Verlag, 1999, pp. 398–416.

[19] R. Bayardo, "Efficiently mining long patterns from databases," in *SIGMOD*. ACM, 1998, pp. 85–93.
[20] T. Calders and B. Goethals, "Non-derivable itemset mining," *Data Min. Knowl. Discov.*, vol. 14, no. 1, pp. 171–206, 2007.
[21] S. Ruggieri, "Frequent regular itemset mining," in *KDD*. ACM, 2010, pp. 263–272.
[22] L. Zhao, M. J. Zaki, and N. Ramakrishnan, "Blosom: A framework for mining arbitrary boolean expressions," in *KDD*. ACM, 2006, pp. 827–832.
[23] B. Bringmann and A. Zimmermann, "The chosen few: On identifying valuable patterns," in *ICDM*. IEEE Computer Society, 2007, pp. 63–72.
[24] B. Bringmann, S. Nijssen, and A. Zimmermann, "Pattern-based classification: A unifying perspective," in *LeGo, 'From Local Patterns to Global Models', Second ECML PKDD Workshop*, 2009.
[25] M. Thoma, H. Cheng, A. Gretton, J. Han, H.-P. Kriegel, A. J. Smola, L. Song, P. S. Yu, X. Yan, and K. M. Borgwardt, "Near-optimal supervised feature selection among frequent subgraphs," in *SDM*. SIAM, 2009, pp. 1075–1086.
[26] H. Cheng, X. Yan, J. Han, and P. S. Yu, "Direct discriminative pattern mining for effective classification," in *ICDE*. IEEE Computer Society, 2008, pp. 169–178.
[27] X. Yan, H. Cheng, J. Han, and D. Xin, "Summarizing itemset patterns: a profile-based approach," in *KDD*. ACM, 2005, pp. 314–323.
[28] F. Afrati, A. Gionis, and H. Mannila, "Approximating a collection of frequent sets," in *KDD*. ACM, 2004, pp. 12–19.
[29] T. Mielikäinen and H. Mannila, "The pattern ordering problem," in *PKDD*. Springer, 2003, pp. 327–338.
[30] A. Gallo, T. Bie, and N. Cristianini, "Mini: Mining informative non-redundant itemsets," in *PKDD*. Springer, 2007, pp. 438–445.
[31] A. Siebes, J. Vreeken, and M. van Leeuwen, "Item sets that compress," in *SDM*. SIAM, 2006.
[32] A. J. Knobbe and E. K. Y. Ho, "Pattern teams," in *PKDD*. Springer, 2006, pp. 577–584.
[33] S. Nijssen, T. Guns, and L. De Raedt, "Correlated itemset mining in roc space: a constraint programming approach," in *KDD*. ACM, 2009, pp. 647–656.
[34] T. Imielinski and A. Virmani, "MSQL: A query language for database mining," *Data Mining and Knowledge Discovery*, vol. 3, pp. 373–408, 1999.
[35] H. Blockeel, T. Calders, É. Fromont, B. Goethals, A. Prado, and C. Robardet, "An inductive database prototype based on virtual mining views," in *KDD*. ACM, 2008, pp. 1061–1064.
[36] F. Bonchi, F. Giannotti, C. Lucchese, S. Orlando, R. Perego, and R. Trasarti, "A constraint-based querying system for exploratory pattern discovery," *Inf. Syst.*, vol. 34, pp. 3–27, 2009.
[37] T. Imielinski and H. Mannila, "A database perspective on knowledge discovery," *Commun. ACM*, vol. 39, pp. 58–64, 1996.
[38] P. V. Hentenryck and L. Michel, *Constraint-Based Local Search*. MIT Press, 2009.
[39] L. D. Raedt and S. Nijssen, "Towards programming languages for machine learning and data mining (extended abstract)," in *ISMIS*. Springer, 2011, pp. 25–32.

Luc De Raedt received his degree in Computer Science from the Katholieke Universiteit Leuven (Belgium) in 1986 and his Ph.D. in Computer Science from the same university in 1991. From 1999 till 2006 he was a full Professor at the Albert-Ludwigs-University Freiburg. Since then, he has taken up a (full) research professorship (BOF) at the Department of Computer Science of the Katholieke Universiteit Leuven (Belgium), where he joined the lab for Declarative Languages and Artificial Intelligence (DTAI). Luc De Raedt's research interests are in Artificial Intelligence, Machine Learning and Data Mining, as well as their applications. His interests are in probabilistic logic learning, the integration of constraint programming with data mining and machine learning principles, the development of programming languages for machine learning, and analyzing graph and network data. He is also interested in applications of these methods to chemo- and bio-informatics, to natural language processing, vision, robotics and action- and activity learning.

Tias Guns received his master degree in Computer Science from the Katholieke Universiteit Leuven (Belgium). He is currently a PhD student at the DTAI lab in Leuven. His research interests include data mining, constraint programming and declarative approaches in general.

Siegfried Nijssen received his master degree and his Ph.D. in Computer Science from the Universiteit Leiden (The Netherlands). He was a postdoc at the Albert-Ludwigs-University Freiburg in 2005-2006, and is a postdoc in the DTAI lab in Leuven since 2006. His research interests include constraint-based data mining, mining patterns in structured data, declarative approaches to data mining and applications in bioinformatics and chemoinformatics.