

# *k*-Pattern Set Mining under Constraints

*Tias Guns, Siegfried Nijssen, Luc De Raedt*

*Report CW596, October 2010*



Katholieke Universiteit Leuven  
Department of Computer Science

Celestijnenlaan 200A – B-3001 Heverlee (Belgium)

# *k*-Pattern Set Mining under Constraints

*Tias Guns, Siegfried Nijssen, Luc De Raedt*

*Report CW 596, October 2010*

Department of Computer Science, K.U.Leuven

## **Abstract**

We introduce the *k*-pattern set mining problem, which is concerned with finding sets of *k* patterns that satisfy constraints. We formulate a number of such constraints, both at the local level, that is, on individual patterns, and more importantly, also on the global level, that is, on the overall pattern set. The resulting framework is flexible and generic in the sense that it can be instantiated to a wide variety of well-known mining tasks including concept-learning, rule-learning, redescription mining, conceptual clustering and tiling. We present a solution method based on constraint programming and discuss how many problems can be modelled in a constraint programming system. Finally, a number of experiments show the promise and generality of the approach.

**Keywords :** 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  $\text{Th}(\mathcal{L}, p, \mathcal{D}) = \{\pi \in \mathcal{L} \mid p(\pi, \mathcal{D}) \text{ is true}\}$ , 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. Despite the popularity of local pattern mining, there are still unsolved problems with this approach.

First, the interaction between the mining and the use of patterns is not well understood. 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 local patterns is typically too large, and that the patterns are too hard to interpret in the global context as they interact with one another. This leads 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. In the second step, some patterns are selected and combined in a heuristic way. One example is associative classification, where systems such as CBA [1] and CMAR [2] build a classifier from a set of association rules. This raises the question as to whether it is possible to develop a *single step* mining approach that solves the *global* pattern set mining problem directly, and avoids having to generate patterns that will never be used anyway.

Secondly, there is a multitude of different global pattern mining methods and approaches, but a general solution is missing. Each combination (or system) is often tailored towards one specific task, such as concept-learning [3], conceptual clustering [4], redescription mining [5], tiling [6], etc. While these combinations are all –to some extent– based on principles of pattern mining, there exists –to the best of the authors’ knowledge– no integrated problem specification, approach or system that is able to tackle all of these tasks in a uniform way. Despite a lot of progress on specific aspects of pattern mining, this hinders our overall understanding of pattern mining. It also raises the question as to whether it is possible to provide a more general approach to data mining that can be instantiated to a wide variety of tasks such as the ones listed above. The answer to this question cannot be expected to be a highly optimized system that is competitive with much more specialized approaches on standard data mining tasks. For many types of data analysis today, run time efficiency is not a major bottleneck anymore. On the other hand, developing novel algorithms and systems to accommodate for new constraints and to tackle novel data mining tasks is still a time consuming task and requires the involvement of highly specialized data mining experts. A general purpose data mining framework and system, in which new problems can be specified in a declarative manner, would alleviate these problems and would be an adequate answer to the above question. It is precisely the goal of this paper to contribute towards such a framework by using constraint programming principles and techniques.

The key contribution of this paper is that we positively answer these two questions. We answer the first question by introducing the problem of mining sets of  $k$  patterns under constraints, that is,  $k$ -pattern sets. A  $k$ -pattern set thus consists of  $k$  patterns and the task is to find those  $k$ -pattern sets that satisfy the constraints. Constraints can be specified both at the local and at the global level that is, both at the level of individual patterns and at the level of the pattern set as a whole. In this paper we focus on patterns in the form of itemsets, which can be regarded as conjunctions of items. A set of patterns is often interpreted as a disjunction of conjunctions, and hence corresponds to a boolean formula in disjunctive normal form (DNF). This paper studies the mining of pattern sets consisting of exactly  $k$  patterns, thus the resulting pattern sets can be conceived as  $k$ -term DNF formulas.

We also provide a positive answer to the second question by showing how instances of the tasks of concept learning, conceptual clustering, tiling, and redescription mining can all be formulated within a constraint programming (CP) framework. Constraint programming (CP) is a generic framework for solving combinatorial and optimization problems under constraints. It has been used successfully in numerous applications including constraint based mining of individual patterns [7, 8]. The key power of constraint programming lies in its declarative approach towards problem solving: in constraint programming, users model a problem by specifying a set of constraints, and the CP system is responsible for solving the problem. In this way, the specification of the problem is separated from the search strategy. This has the advantage that different problems can be specified by merely changing the declarative problem specification in terms of constraints. It is this approach that we will pursue in the present paper. By casting data mining

problems in constraint programming models, we will show that a wide variety of problem settings can be addressed in a uniform way.

An important part of this work is the extension of our earlier work on constraint programming for local pattern mining [7, 8] towards the mining of entire pattern sets. Our earlier work showed that appropriate constraint programming models can lead to practical solutions for computationally hard problems, in particular for the case of correlated itemset mining in [8]. The challenge is here to identify the modelling choices that make the approach also for pattern set mining feasible. Previous studies made a distinction between constraints on the local level (on individual patterns) and the global level (on patterns sets as a whole). We will identify two special cases, namely, local look-ahead constraints and global pairwise constraints. Such constraints are more powerful than their regular counterparts, making them crucial to reduce the search space to manageable size. This categorisation also gives us new insights into the relationship between local and global constraints.

The resulting framework for pattern set mining is very flexible and allows us to specify a wide range of problems for a wide range of data mining tasks. Examples of constraints are that patterns must be frequent, that the pairwise overlap between patterns is small, that the pattern set occurs frequently in a set of positive examples and is highly infrequent in a set of negative examples, or that the coverage of the overall pattern set in a data set is maximised.

The experiments show clearly the generality and flexibility of the constraint programming approach to  $k$ -pattern set mining. They show that the constraint programming approach can find solutions for many  $k$ -pattern set mining problems of interest, especially in those cases where the solutions are heavily constrained. On the other hand, because CP is based on exhaustive search and guarantees to find a global optimum, solving problems with few, weak constraints is often not tractable. In practice, this means that even though many problems can be modelled, the solution strategy of existing CP systems is only feasible for certain types of models. Nevertheless, also in such cases the formulation as a constraint programming model provides insights that may be used for developing more heuristic declarative approaches to the  $k$ -pattern set mining problem (e.g., using local search), and hence this work may contribute to the longer term vision of developing general purpose declarative data mining tools. This is similar to the constraint programming approach to mining for local patterns [7].

While several approaches to finding pattern sets are described in the literature, cf. [9] and Section 2, there exists – to the best of the authors’ knowledge – no single step approach to the global pattern set mining problem, nor has any technique been applied to such a wide range of data mining tasks as the one that we shall describe here. A detailed discussion of related work is, however, deferred to Section 6.

## 2 $k$ –Pattern Set Mining

In general, pattern mining is concerned with finding all patterns  $\pi$  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  $\mathcal{D}$ :

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

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 local itemset mining problem is that of finding  $\text{Th}(\mathcal{I}, p, \mathcal{D}) = \{I \subseteq \mathcal{I} \mid p(I, \mathcal{D}) \text{ is true}\}$ . The pattern language is the entire 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

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

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  $\text{Th}(\mathcal{L}, p, \mathcal{D})$  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:

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

The pattern set  $\Pi$  consists of  $k$  patterns  $\pi$ . In its whole, the pattern set  $\Pi$  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  $\Pi$  by searching, for example, for the one with maximum  $f(\Pi)$ :

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

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  $\Pi$  consists of  $k$  individual itemset patterns  $\pi$ . Every pattern  $\pi$  is represented by its itemset  $I$  and transaction set  $T$ :  $\pi = (I, T)$ . We write  $\Pi = (\pi^1, \dots, \pi^k) = ((I^1, T^1), \dots, (I^k, T^k))$ . 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 five 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 in 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. The *canonical form constraints* at last are used to order the patterns within the pattern set. This enforces a canonical form on the pattern set, which avoids finding identical pattern sets with different syntactical forms.

### 2.1.1 Individual Pattern Constraints

These are the constraints that have been identified in the framework of constraint-based mining [7]. We will review some of the most important constraints for the itemset mining problem here.

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

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

*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.

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

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

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

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  $\leq$  we will denote any comparison  $\leq \in \{<, \leq, >, \geq, =, \neq\}$ . A constraint on the size of the pattern can thus be formulated as

$$size(\pi) \leq \theta : |I| \leq \theta. \quad (9)$$

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

$$freq(\pi) \leq \theta : |T| \leq \theta. \quad (10)$$

### 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.

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

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

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.

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

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

Such measures can be constrained by a comparison operator  $\leq \in \{<, \leq, >, \geq, =, \neq\}$  and a threshold  $\theta$ .

*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:

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

An alternative approach is to bound the minimum or maximum value over all evaluations:

$min(dist(\pi^1, \pi^2), dist(\pi^1, \pi^3), \dots, dist(\pi^2, \pi^3), \dots) \leq \theta$ . 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  $\theta$ , this can be rewritten as follows:

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

### 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  $\Pi$  is not explicit in our formulation, but can be deduced from the covers of all the 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:

$$cover(\Pi) = T^\Pi = T^1 \cup \dots \cup T^k. \quad (17)$$

*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:

$$freq(\Pi) \leq \theta : |T^\Pi| \leq \theta. \quad (18)$$

*Area of pattern set.* The area of a pattern set was studied in the context of large tile mining [10]. 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 area of all the tiles of the individual patterns. Note that tiles can be overlapping, but every tuple  $(t, i)$  covered is only counted once.

$$area(\Pi) = |tile(\pi^1) \cup \dots \cup tile(\pi^k)|. \quad (19)$$

#### 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  $T^\Pi$  is defined as in equation (17):

$$cover^+(\Pi) = T^\Pi \cap \mathcal{T}^+, \quad (20)$$

$$cover^-(\Pi) = T^\Pi \cap \mathcal{T}^-. \quad (21)$$

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  $N = |\mathcal{T}^-|$ . To calculate the positive/negative examples covered by an entire pattern set we replace  $cover^+(\pi)$  by  $cover^+(\Pi)$  and  $cover^-(\pi)$  by  $cover^-(\Pi)$ :

$$p = freq^+(\Pi) = |cover^+(\Pi)| = |T^\Pi \cap \mathcal{T}^+|, \quad (22)$$

$$n = freq^-(\Pi) = |cover^-(\Pi)| = |T^\Pi \cap \mathcal{T}^-|. \quad (23)$$

*Accuracy of a pattern set.* Accuracy is defined as the proportion of positive examples covered:  $(p + (N - n))/(P + N)$ . As  $P$  and  $N$  are constant for a given dataset, one can equivalently optimize the formula  $p - n$  [11]. Using equations (22) and (23) and the principles explained above we get the following formulation:

$$accuracy(freq^+(\Pi), freq^-(\Pi)) = freq^+(\Pi) - freq^-(\Pi). \quad (24)$$

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

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

$$Laplace(freq^+(\Pi), freq^-(\Pi)) = \frac{freq^+(\Pi) + 1}{freq^+(\Pi) + freq^-(\Pi) + 2}. \quad (26)$$

The topic of identifying such measures has already been studied extensively in pattern mining [12, 1] and in rule learning [11].

### 2.1.5 Canonical Form Constraints

An important issue in many pattern mining tasks is how to avoid syntactical variations of the same pattern from being generated. In pattern set mining we also face this problem. For instance, the following two pattern sets can be considered equivalent:  $(\pi_1, \pi_2)$  and  $(\pi_2, \pi_1)$ . To ensure that only one of these representations of the pattern is found, we will impose a canonical form on the pattern set by means of a constraint.

The details of the constraints that can be used to enforce a canonical form will be discussed in the context of our Constraint Programming implementation.

## 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. For the optimisation problems, the value or function to optimise will be indicated by the maximise or minimise keywords.

### 2.2.1 $k$ -term DNF Learning and Concept Learning

The main aim of  $k$ -term DNF learning 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 formalise this problem as finding pattern sets  $\Pi$  satisfying:

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

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 in the formalisation; the main motivation for this choice is that for any  $k$ -term DNF containing non-closed itemsets, we can find a corresponding  $k$ -term DNF with only closed itemsets; hence, non-closed itemsets can be considered redundant. Adding this constraint 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:

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

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:

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

In all cases, the result will be a  $k$ -pattern set with high accuracy on the examples. Every pattern  $\pi$  will represent 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  $\Pi$  that do not overlap and cover all the examples:

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

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:

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

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:

$$\underset{\forall \pi^i \in \Pi}{\text{minimise}} \quad \max(\text{freq}(\pi^1), \dots, \text{freq}(\pi^k)) - \min(\text{freq}(\pi^1), \dots, \text{freq}(\pi^k));$$

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:

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

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 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. However, in preliminary experiments we found that successfully modelling this type of problem will require optimizations that are beyond the scope of this work.

### 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:

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

This models the original setting in which zeros can not be covered by a pattern. We could model the *fault-tolerant* setting as well, but solving it would become significantly harder.

#### 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  $\mathcal{I}^1 \dots \mathcal{I}^k$  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  $k$  where for every pattern  $\pi^p = (I^p, T^p) : I^p \subseteq \mathcal{I}^p$  and:

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

In this case all patterns have to cover exactly the same transactions, which may be a too stringent requirement. Instead, one could also solve the following problem:

$$\begin{aligned} & \underset{\Pi}{\text{minimise}} && \text{sumdist}(\Pi), \\ & \forall \pi \in \Pi : && \text{covered}(\pi), \\ & \forall \pi \in \Pi : && \text{closed}(\pi), \\ & \forall \pi \in \Pi : && \text{freq}(\pi) \geq \theta, \end{aligned}$$

where we search for an accurate redescription of a sufficiently large number of examples. 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 propose to use constraint programming (CP) as a technique to solve  $k$ -pattern set mining problems. We will 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.

---

**Algorithm 1** Constraint-Search( $D$ )

---

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

---

**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 if either the mother or the grandfather does. 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. We will specify the constraints by summing over these boolean variables. In line (31) below, 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 only iff either the mother or grandfather joins.

$$D(G), D(F), D(M), D(Ch_1), D(Ch_2) = \{0, 1\} \quad (28)$$

$$F + M \geq 1 \quad (29)$$

$$F \leftrightarrow M + G = 1 \quad (30)$$

$$G + F + M + Ch_1 + Ch_2 = 3 \quad (31)$$

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 constraint is implemented by a propagator. The constraints we will use are often of the form  $Function(V) \leq X$  where  $\leq \in \{<, \leq, >, \geq, =, \neq\}$ ,  $V$  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:

---

**Algorithm 2** Propagator for  $\sum \vec{B} \geq \theta$ 


---

```

1: Lowerbound :=  $\sum_{B \in \vec{B}} \min D(B)$ 
2: Upperbound :=  $\sum_{B \in \vec{B}} \max D(B)$ 
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  $B \in \vec{B} : D(B) = \{\max D(B)\}$  % Propagate
11: end if

```

---

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  $Function(V) \leq X$  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  $\vec{B} = (B_1, B_2, \dots, B_n)$ . A propagator for the constraint  $\sum_{i=1}^n B_i \geq \theta$ , abbreviated by  $\sum \vec{B}$ , 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  $\leq$  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 [13].

Another important concept that we will often use is the reification of constraints. A reified constraint is one that binds a boolean variable to the *truth value* of the constraint. Consider the reified sum constraint  $B \leftrightarrow V_1 + V_2 \geq \theta$ . The effect of this constraint is that whenever the truth-value of  $V_1 + V_2 \geq \theta$  is known, this can be propagated to  $B$ . Vice versa, whenever the value of  $B$  is known, the (un)satisfaction of  $V_1 + V_2 \geq \theta$  can be propagated. For instance, suppose we know that  $B = 0$ , then the constraint  $V_1 + V_2 < \theta$  will be posted and take effect. Similar to reification it is possible to imply-reify a constraint:  $B \rightarrow V_1 + V_2 \geq \theta$ , where the truth value of  $B$  implies the satisfaction of the constraint.

In many of our models, we use reified boolean variables that are only used in arithmetic constraints. 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  $\vec{V} = V_1 \dots V_n$  and  $\vec{W} = W_1 \dots W_n$  of equal length  $n$ , and we wish to sum over the truth values  $B_x$  where  $\forall x \in \{1 \dots n\} : B_x \leftrightarrow V_x + W_x \geq 1$ . We can abbreviate this sum to  $\sum_x [V_x + W_x \geq 1]$ .

Note that we can implement special propagators for summations over Iverson brackets if we wish to avoid that the CP system has to explicitly maintain the reified boolean variables.

**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*

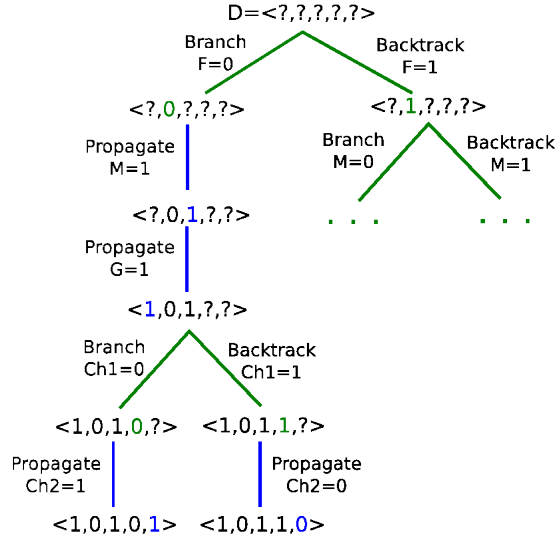


Figure 1: Search tree for the holiday example.

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 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 procedure 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

Given the principles of constraint programming discussed in the previous section, the question is now how the problem specifications in section 2 can be expressed and effectively solved in constraint programming. We first study how the individual constraints can be modelled in a constraint programming framework, and how effective their propagation will be. Next, we take a closer look at how the problem instantiations are modelled in the framework.

### 4.1 Modelling Constraints

Given the principles of constraint programming discussed in the previous section, the question is now how the problem specifications in section 2 can be expressed and effectively solved in constraint programming.

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

Table 1: Individual pattern constraints

While reviewing each of the constraint families, we will categorise them. 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 non-fixed number of variables. 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. 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 identify local constraints by being monotone or anti-monotone with respect to set inclusion, common in constraint-based mining for many years. 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:

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

To constrain the frequency, we could simply constrain this sum, for example given a minimal frequency

threshold  $\theta$ :  $\sum_{t \in \mathcal{T}} T_t \geq \theta$ . 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:

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

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).

*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

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

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

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

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.

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

Table 2: Redundancy constraints and their combinations

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

Table 3: Coverage and discriminative 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:

$$\sum_{a < b} dist(\pi^a, \pi^b) \leq \theta. \quad (36)$$

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  $\theta$ :

$$\left( \min_{a < b} dist(\pi^a, \pi^b) \right) \geq \theta. \quad (37)$$

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 have a number of independent pairwise constraints instead of one global constraint on many pairwise distances. 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  $T^\Pi = T^1 \cup \dots \cup T^k$ . If one pattern covers a transaction, then the pattern set also covers it. In constraint programming we can model this by introducing a temporary variable  $B_t$  for every transaction  $t$ , where  $B_t \leftrightarrow (\sum_{p \in \{1..k\}} T_t^p) \geq 1$ . 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. In this case,  $|\mathcal{I}| * |\mathcal{T}|$  temporary variables are needed to represent each cell of the matrix individually. Using existing summation constraints, we can expect this constraint to propagate worse than other global constraints.

### 4.1.4 Canonical Form Constraints

The pattern set has to be constrained to a canonical form to avoid finding duplicate solutions. Such a canonical form can be enforced by imposing a strict ordering on the patterns in the pattern set. In the constraint programming community the lexicographic ordering constraint is commonly used to achieve this and its propagation has hence been study extensively [13]. Instead of imposing a lexicographic ordering on the structure of the patterns, one could also order the patterns primarily on some other property, like its frequency or accuracy, and use a lexicographic order on the structure of the patterns to break ties. Given that the order constraint can be enforced between every pair of patterns, this constraint falls in the category of *pairwise constraints*.

$$\text{canonical}(\Pi) : \pi^1 < \pi^2 < \dots < \pi^k \quad (38)$$

There are some design decisions to be made when ordering patterns. 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. 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. When we fill in the formulas of the previous section in the model of equation (27), we obtain the following CP model:

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

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

$$\forall t \in \mathcal{T} : B_t = \left[ \left( \sum_{p \in \{1..k\}} T_t^p \geq 1 \right) \right],$$

$$\sum_{t \in \mathcal{T}^+} B_t - \sum_{t \in \mathcal{T}^-} B_t \geq \theta, \quad (\text{Accurate})$$

$$T^1 < T^2 < \dots < T^k \quad (\text{Canonical})$$

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. Nevertheless, we would like to point out that one can also use CP systems to find an optimal solution in two steps. In this case, we first search for all itemsets that fulfil the local constraints:

$$\begin{aligned} & coverage(\pi), \\ & closed(\pi), \end{aligned}$$

In this model we only search for one pattern at a time. Using all itemsets found, we create a new transactional database, in which every item represents an itemset. On this new transactional database we solve another pattern mining problem with the following constraints:

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

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  $\theta$ . This approach is similar to the approach chosen in [9], 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.

## 5 Experiments

The key contribution of this paper is the introduction of a novel and flexible framework for mining  $k$ -pattern sets. As stated before, we focus on generality and flexibility rather than efficiency or scalability. Therefore, it cannot be expected that our technique would be more efficient or more scalable than the more specialized and optimized data mining algorithms that often focus on a single task only and perform heuristic rather than exhaustive search. The purpose of the experiments is hence rather to obtain insight in the problem of  $k$ -pattern set mining problem, its possibilities, limitations and challenges for further research.

The experimental section is therefore intended to answer 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. can the categorization of the constraints (into global, pairwise, local and local look-ahead) be used to characterize the performance of the algorithm?

We will study these questions by performing experiments for each of the four tasks of interest, that is, concept-learning, clustering, tiling and redescription mining. We then also formulate 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. For constraints of the form  $\sum[B^1 + B^2 \leq \alpha]$  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.

### 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.

	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

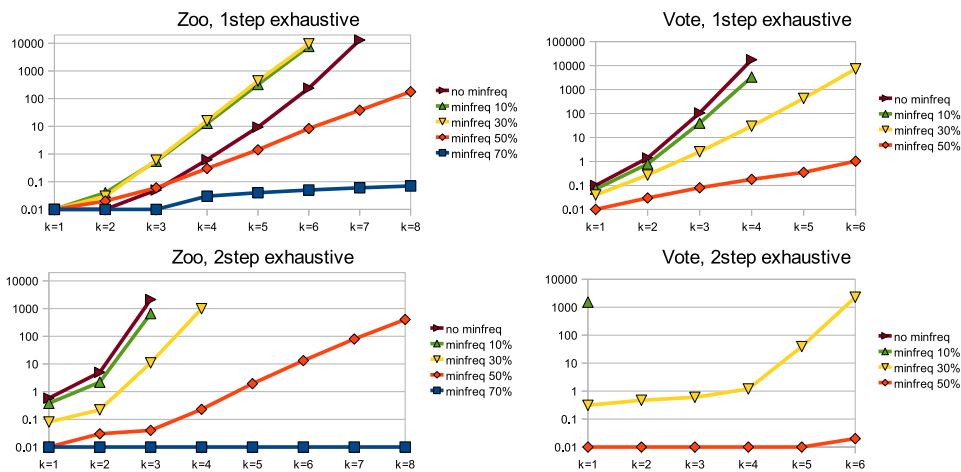


Figure 2: Comparing a one step exhaustive search with a two step exhaustive search.

In our first experiment, we investigate the first question: how does the proposed one step approach compare to the two step procedure (Q1)? In the two step approach, first all patterns given a minimum frequency threshold are mined, and in the second step the best combination of  $k$  patterns is sought. In the one step approach we also impose the minimum frequency constraint, but search for the  $k$  pattern set directly, as indicated in Section 4.2. Figure 2 shows the result of this experiment for the zoo and vote datasets, using different frequency thresholds. Our proposed one step approach (top row) is faster and scales to higher  $k$  than the two step approach in most cases, especially for low frequency thresholds. The two step approach generally suffers from the large amount of candidate patterns that its first step generates. The one step approach on the other hand is able to effectively use all the constraints, thereby making it possible to find the optimal  $k$ -pattern set up to certain values of  $k$ , even without a frequency constraint.

Hence, on this task we can conclude that it is beneficial to search for  $k$ -pattern sets directly.

Next, we investigate the question how the approach scales to different tasks (Q2). In Figure 3 we see 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 [7], as well as the global accuracy constraint and pairwise lexicographic ordering constraints (we still use no frequency constraint here). 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

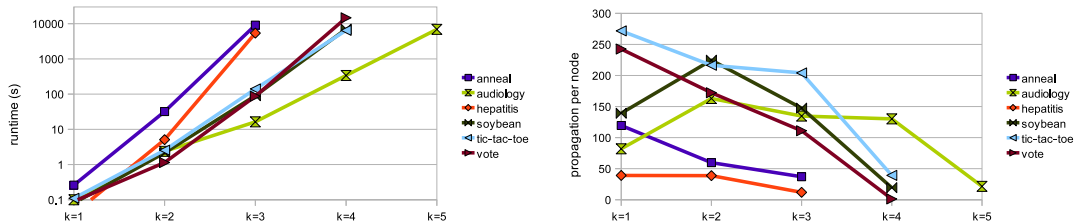


Figure 3: Runtime and number of propagations per node for the concept learning setting.

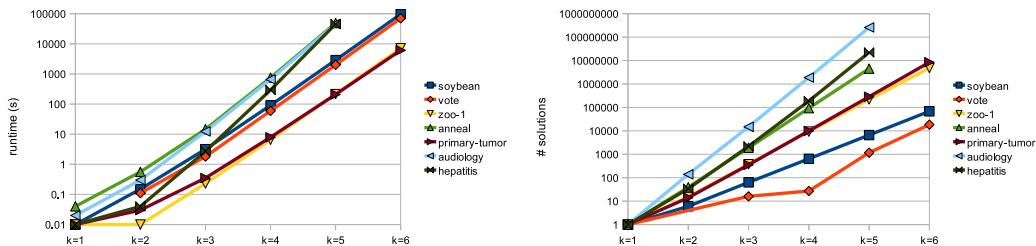


Figure 4: Runtime and number of conceptual clusterings for varying  $k$

patterns. Starting from  $k = 3$  more pairwise constraints are added, and the accuracy constraint becomes a truly 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 from  $k = 3$  on, the number of propagations per node quickly decreases. Knowing that the number of nodes in the search tree grows quickly for increasing  $k$ , it becomes clear that with such low amounts of propagation per node, the search can not efficiently find the optimal solution. Clearly, for larger  $k$  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. 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: loose constraints, in our case a high value of  $k$ , leads to a combinatorial explosion where most time is spent on enumerating the solutions.

When we studied the resulting clusterings in more detailed, 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.

Figure 5a illustrates the first optimisation setting, in which we maximise the minimum cluster size, while Figure 5b illustrates the second setting, in which we minimise the cluster range. In both cases, the

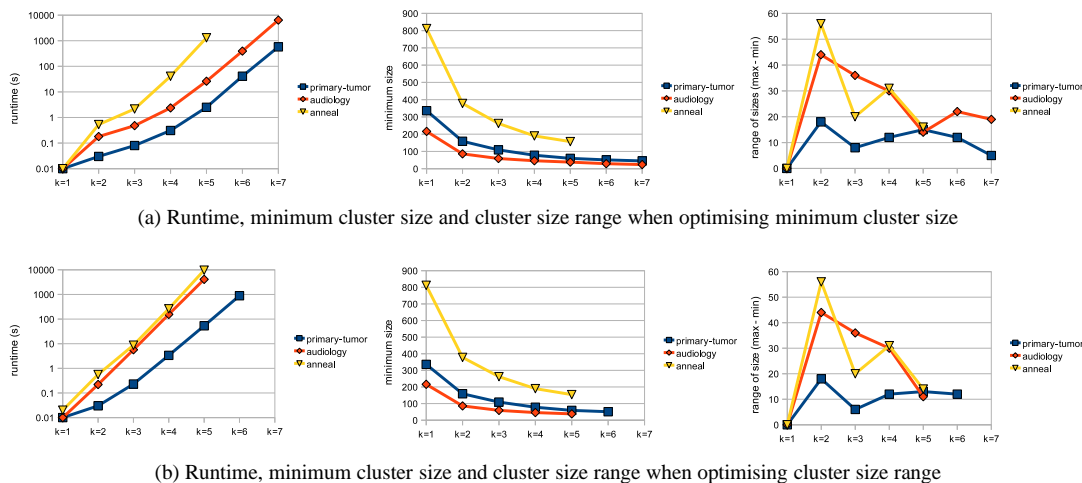


Figure 5: Mining a pattern set clustering with  $k$  conceptual clusters.

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 Figure 5a and Figure 5b. 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 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 our model of the  $k$ -tiling problem, we determine the area of a tiling by summing variables for all elements in the data matrix. This means that the problem is mostly constrained by a global optimisation criterion, which acts as a global constraint during the search. If we consider how the propagation bounds the search for this model, the area of a  $k$ -tiling is bounded by the sum of all elements in the database that can be covered by at least one tile. This means that in most cases the entire pattern set has to be set by the search, before the optimistic estimate of the area will be low enough to be used by the branch-and-bound search. Hence, one could describe our model of the  $area(\Pi)$  constraint to be very loose. For this reason, we do not expect the  $k$ -tiling model to perform well, and a good test to investigate the limits of our CP approach further (Q2).

Table 5 shows the maximum  $k$  for which a  $k$ -tiling was found within a time limit of 6 hours per run. We also report the area of that  $k$ -tiling, and the area found by a greedy  $k$ -tiling algorithm for the same value of  $k$ . The results confirm our expectation: only for very low  $k$  can the optimal  $k$ -tiling be found. For values of  $k$  of 3 and higher, the area constraint is too weak to prune the search space significantly, and the search space is too big to enumerate exhaustively. When an optimal  $k$ -tiling is found it is often, but not always, better than the  $k$ -tiling found using the greedy search approach.

	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.

## 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. We are interested in finding redescrptions, 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 redescrptions. In the second approach, we first find all closed itemsets that have at least one item in each partition. Redescrptions 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 distinctiveness of zero corresponds to an exact redescription. The run times for finding the redescrptions are generally low in these settings. Except for the zoo dataset, the relative frequency of the redescrptions in Table 6 is low (column 3). For these datasets, there are no exact redescrptions 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 redescrptions, 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 Conclusions of the Experiments

When we compare the results on the different experiments, we can draw the following conclusions.

	runtime (s)	relative frequency	runtime closed (s)	number 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 several settings of redescription mining; 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.

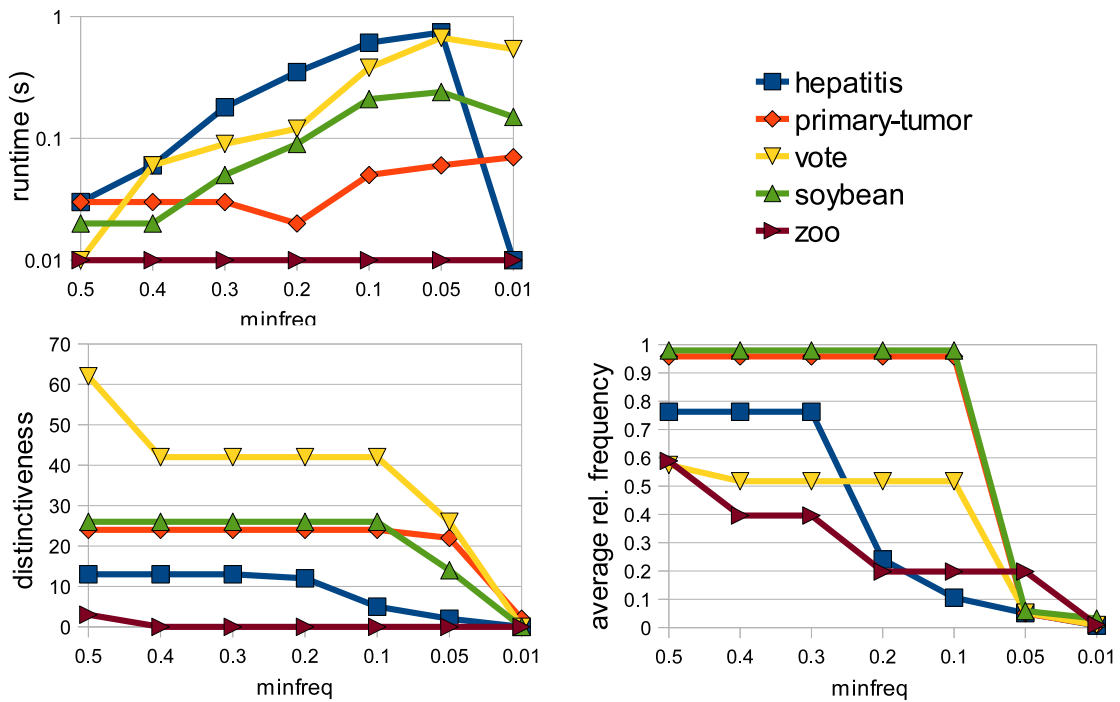


Figure 6: Run time, distinctiveness and average frequency of patterns when searching the least distinct redescription given a minimum frequency threshold.

We evaluated on the problem of concept learning whether it is beneficial to use a one step approach as compared to a two step approach for finding optimal pattern sets (Q1). These results indicate that the one step approach can indeed find identical solutions with shorter run times; whereas in the two step approach we are faced with the bottleneck of searching over subsets of large numbers of patterns, in the one step approach we can use the constraints of the global problem to reduce the number of patterns that need to be considered.

When we compare the different tasks with each other (Q2), we can observe that there are significant differences in the scalability of the approach to different tasks. Sorting these tasks from easy to difficult, we find that our model of redescription mining can easily be solved; concept learning and conceptual clustering are harder, while tiling is not tractable even on some of the smaller datasets. For the tasks of concept learning and conceptual clustering, we evaluated several different modelling choices. The impact of these choices was in particular clear on the problem of conceptual clustering, where we showed that two different models in practice found the same solutions, but had quite different run times.

The main reason for the differences in run time are the types of constraints that are used (Q3). By restricting our setting of redescription mining to sets of two patterns, we only have local constraints and pairwise constraints in this problem; we found that the problem is consequently relatively easily solved. In

our model of tiling, on the other hand, we had few additional constraints next to the coverage and closedness constraints. Propagation here turned almost impossible for  $k$  larger than two. On other problems, such as conceptual clustering, it was beneficial to formulate the problem in such a way that we effectively obtain a minimum frequency constraint during the search. This confirmed the overall importance of imposing sufficient local constraints.

The general lessons that can be drawn from our experiments are hence that it is beneficial to formulate the  $k$  pattern set mining problem as a one step problem if one is interested in finding optimal solutions, but that the feasibility of finding such optimal sets is dependent on the problem; one should aim at models with sufficient local constraints and pairwise constraints to reduce the search space.

## 6 Related Work

Compared to 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 in which first a set of local patterns is constructed, and in a second step the pattern set is extracted.

It is nevertheless useful to look into alternative techniques for removing redundancy in the output of local pattern mining algorithms, as this problem has been generally recognised ([16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 9]). These techniques can be distinguished on whether they remove redundancy at the local level, that is, at the level of individual patterns, or at the global level by constructing a concise set of patterns. We now discuss each of these techniques in turn.

### 6.1 Local Techniques

Techniques for removing redundancy at the local level typically focus on finding patterns that form a condensed representation ([16, 17, 18]). Condensed representations range from exact representations for which the frequency of each pattern can be reconstructed, to lossy representations that discard a part or all information regarding frequency [18]. 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 many two step algorithms. Our approach considers only closed frequent patterns. It is possible use other condensed representations such as the ones defined in [7], although it could be that the globally optimal pattern set does not exist for some of the 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 [9]. 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 [9] 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 BLOSUM framework [27], which can mine for all DNF and CNF expressions in a binary dataset. BLOSUM 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 Two Step Techniques

While [9] 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, 28]) (see [29] 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, [28, 30]) define measures of redundancy and ways to select only a limited number of patterns. 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 [22], finding the patterns that best approximate the entire collection [21], ordering the patterns such that each prefix is a good summary [20], ordering according to statistical p-value [19], 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 [25] 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 [21] could be applied on the dataset. However, in both cases, again a heuristic two step algorithm is used and the size of the selected pattern set is unbounded.

In [24], 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 [8] 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 [24] 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.

## 7 Conclusions

We have introduced the problem of  $k$ -pattern set mining and showed how it can be instantiated in many different ways to solve a wide variety of different tasks. Following the paradigm of constraint programming, we have presented a framework in which problems are defined through different types of constraints. By simply varying the constraints, it is possible to formulate different problem settings and tasks. The range of tasks that can be expressed includes instances of concept learning, clustering, tiling and redescription mining. This shows the flexibility and generality of the approach, certainly as compared to existing approaches to data mining which often focus on a single task and only a few constraints.

By identifying five families of constraints, it is possible to formulate different problem settings and tasks by simply varying these constraints. One of the biggest benefits is that one does not have to stick to one problem formulation. Instead, other constraints could be added or replaced, resulting in different variants. The need for different problem formulations could be driven by the application at hand, or by the computational effects of choosing certain constraints.

We demonstrated how the constraints can be formulated as a CSP, and how standard constraint programming solvers can subsequently search for the optimal  $k$ -pattern set. The feasibility of this approach was proven in a number of experiments. The experiments also showed which problem instantiations, and more specifically, which combination of constraints, can reduce the huge search spaces to manageable sizes. In this regard, we identified five families of constraints that typically differ in the amount of propagation they allow and hence influence the efficiency of the search.

By studying the local and global constraints based on their propagation power, we have contributed to a better understanding of the relationships between local and global constraints, especially with the identification of pairwise constraints as a kind of global constraint with a propagation power that is more similar to that of local constraints. A further study on what constraints can be decomposed into pairwise constraints could lead to advances, not only for one step exhaustive search methods, but also for two step approaches. In general, the study of the relation between local and global constraints, not necessarily in a one step framework, merits further study.

There are several other interesting issues for future work. First, although we have shown that reasonable results can be obtained for some problem settings, others were harder to tackle within our framework. The question remains open as to whether optimal pattern sets for those problems can be found using alternative propagators that take into account additional properties of the constraints in data mining. Furthermore, the scalability of the approach to larger values of  $k$  and larger datasets could also be investigated. Second, while we now solve the  $k$ -pattern set mining problems using exhaustive search methods, it would be interesting to investigate whether the constraints could also be used with, for example, local-search methods. Even though this would result in approximate solutions, it could also lead to a better understanding and a more general perspective on pattern set mining.

Further variations and combinations of the constraints used throughout this paper might lead to the formulation of novel data mining tasks. The framework can – in this regard – serve both for rapid prototyping and as a benchmark when developing new algorithms.

This framework implements a part of our vision to develop general, declarative data mining tools. Possibilities for future work include its extension towards more general clustering tasks, 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 implementing a series of models in an integrated solver environment.

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.

## Acknowledgements

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*, 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*, N. Cercone, T. Y. Lin, and X. Wu, Eds. IEEE Computer Society, 2001, pp. 369–376.
- [3] M. Kearns and U. Vazirani, *An introduction to computational learning theory*, M. T. M. P. Cambridge, Ed., 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*, M. M. Veloso and S. Kambhampati, Eds. AAAI Press / The MIT Press, 2005, pp. 837–844.
- [6] F. Geerts, B. Goethals, and T. Mielikäinen, “Tiling databases,” in *Discovery Science*, ser. Lecture Notes in Computer Science, E. Suzuki and S. Arikawa, Eds., vol. 3245. Springer, 2004, pp. 278–289.
- [7] L. De Raedt, T. Guns, and S. Nijssen, “Constraint programming for itemset mining,” in *KDD*, 2008, pp. 204–212.
- [8] S. Nijssen, T. Guns, and L. De Raedt, “Correlated itemset mining in roc space: a constraint programming approach,” in *KDD '09: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. New York, NY, USA: ACM, 2009, pp. 647–656.
- [9] L. De Raedt and A. Zimmermann, “Constraint-based pattern set mining,” in *SDM*. SIAM, 2007.
- [10] F. Geerts, B. Goethals, and T. Mielikäinen, “Tiling databases,” in *Discovery Science*. Springer, 2004, pp. 278–289.
- [11] 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.
- [12] S. Morishita and J. Sese, “Traversing itemset lattice with statistical metric pruning,” in *PODS*. ACM, 2000, pp. 226–236.
- [13] F. Rossi, P. van Beek, and T. Walsh, *Handbook of Constraint Programming*. Elsevier, 2006.
- [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] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal, “Discovering frequent closed itemsets for association rules,” in *ICDT '99: Proceedings of the 7th International Conference on Database Theory*. London, UK: Springer-Verlag, 1999, pp. 398–416.
- [17] R. J. Bayardo, Jr., “Efficiently mining long patterns from databases,” in *SIGMOD '98: Proceedings of the 1998 ACM SIGMOD international conference on Management of data*. New York, NY, USA: ACM, 1998, pp. 85–93.
- [18] T. Calders and B. Goethals, “Non-derivable itemset mining,” *Data Min. Knowl. Discov.*, vol. 14, no. 1, pp. 171–206, 2007.
- [19] A. Gallo, T. Bie, and N. Cristianini, “Mini: Mining informative non-redundant itemsets,” in *PKDD 2007: Proceedings of the 11th European conference on Principles and Practice of Knowledge Discovery in Databases*. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 438–445.
- [20] T. Mielikäinen and H. Mannila, “The pattern ordering problem,” in *PKDD*, ser. Lecture Notes in Computer Science, N. Lavrac, D. Gamberger, H. Blockeel, and L. Todorovski, Eds., vol. 2838. Springer, 2003, pp. 327–338.
- [21] F. Afrati, A. Gionis, and H. Mannila, “Approximating a collection of frequent sets,” in *KDD '04: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. New York, NY, USA: ACM, 2004, pp. 12–19.

- [22] X. Yan, H. Cheng, J. Han, and D. Xin, “Summarizing itemset patterns: a profile-based approach,” in *KDD '05: Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*. New York, NY, USA: ACM, 2005, pp. 314–323.
- [23] A. J. Knobbe and E. K. Y. Ho, “Maximally informative k-itemsets and their efficient discovery,” in *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. New York, NY, USA: ACM, 2006, pp. 237–244.
- [24] ———, “Pattern teams,” in *PKDD*, ser. Lecture Notes in Computer Science, J. Fürnkranz, T. Scheffer, and M. Spiliopoulou, Eds., vol. 4213. Springer, 2006, pp. 577–584.
- [25] A. Siebes, J. Vreeken, and M. van Leeuwen, “Item sets that compress,” in *SDM*, J. Ghosh, D. Lambert, D. B. Skillicorn, and J. Srivastava, Eds. SIAM, 2006.
- [26] F. Pennerath and A. Napoli, “The model of most informative patterns and its application to knowledge extraction from graph databases,” in *ECML PKDD '09: Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases*. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 205–220.
- [27] L. Zhao, M. J. Zaki, and N. Ramakrishnan, “Blossom: A framework for mining arbitrary boolean expressions,” in *In: Proceedings of the 12th International Conference on Knowledge Discovery and Data Mining (KDD 2006)*, 2006, pp. 827–832.
- [28] B. Bringmann and A. Zimmermann, “The chosen few: On identifying valuable patterns,” in *ICDM*. IEEE Computer Society, 2007, pp. 63–72.
- [29] B. Bringmann, S. Nijssen, and A. Zimmermann, “Pattern-based classification: A unifying perspective,” in *Proceedings of 'From Local Patterns to Global Models': Second ECML PKDD Workshop (LeGo)*, A. Knobbe and J. Frnkranz, Eds., 2009.
- [30] 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.