HAL
open science

Integer Linear Programming for Pattern Set Mining; with an Application to Tiling

Abdelkader Ouali, Albrecht Zimmermann, Samir Loudni, Yahia Lebbah, Bruno Crémilleux, Patrice Boizumault, Lakhdar Loukil

To cite this version:

Abdelkader Ouali, Albrecht Zimmermann, Samir Loudni, Yahia Lebbah, Bruno Crémilleux, et al.. Integer Linear Programming for Pattern Set Mining; with an Application to Tiling. Pacific-Asia Conference on Knowledge Discovery and Data Mining, May 2017, Jeju, South Korea. 10.1007/978-3-319-57529-2_23 . hal-01597819

HAL Id: hal-01597819

https://hal.science/hal-01597819v1

Submitted on 28 Sep 2017

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

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Integer Linear Programming for Pattern Set Mining; with an Application to Tiling

Abdelkader Ouali[1,2], Albrecht Zimmermann[2], Samir Loudni[2], Yahia Lebbah[1], Bruno Cremilleux[2], Patrice Boizumault[2], and Lakhdar Loukil[1]

[1] Lab. LITIO, University of Oran 1, 31000 Oran, Algeria.
[2] Normandie Univ, UNICAEN, ENSICAEN, CNRS, GREYC, 14000 Caen, France.

Abstract. Pattern set mining is an important part of a number of data mining tasks such as classification, clustering, database tiling, or pattern summarization. Efficiently mining pattern sets is a highly challenging task and most approaches use heuristic strategies. In this paper, we formulate the pattern set mining problem as an optimization task, ensuring that the produced solution is the best one from the entire search space. We propose a method based on integer linear programming (ILP) that is exhaustive, declarative and optimal. ILP solvers can exploit different constraint types to restrict the search space, and can use any pattern set measure (or combination thereof) as an objective function, allowing the user to focus on the optimal result. We illustrate and show the efficiency of our method by applying it to the tiling problem.

1 Introduction

Pattern mining is one of the fundamental tasks in the process of knowledge discovery, and a range of techniques have been developed for producing extensive collections of patterns. However, resulting pattern collections are generally too large, difficult to exploit, and unstructured – without interpretable relations between patterns. This explains the interest of the community to *pattern sets* [18]. Instead of evaluating and selecting patterns individually, pattern sets (i.e., sets of patterns) assemble local patterns to provide knowledge from a high-level viewpoint, using quality measures that evaluate, and constraints that constrain, the entire set. Examples of problems related to pattern sets include concept-learning, database tiling, data compression, or clustering, to cite a few. However, as the number of possible pattern sets is exponential in the size of the set of local patterns, which is itself huge, the computational efficiency of pattern set mining is a very challenging task. For specific quality measures, such as joint entropy, relatively tight upper bounds can be derived to prune candidate sets [13]. Unfortunately, such pruning strategies are limited to very few cases. In practice, most approaches use a step-wise strategy in which first all local patterns are computed, then heuristically post-processed according to an objective function to be optimized. Therefore only a *single* pattern set is returned; examples are [2,4,14,20]. Obviously this process does not ensure the optimality of the returned pattern set according to the objective function. To the best of our knowledge, only the algorithm proposed in [18] proceeds by exhaustive search while pruning parts of the search space by using pattern set constraints. However, these pruning effects can be weak, and the number of patterns being very large, this method only works for small pattern collections.

In this paper, we formulate the pattern set mining problem as an optimization task, ensuring that the produced solution is the best from the entire search space. In a sense, we return to the spirit of the original idea of pattern set mining [18] based on a complete method. However, we produce only the best solution, avoiding being drowned by patterns. What's more, we use constraint programming (CP) techniques since CP solvers can exploit a wider range of constraints than data mining approaches that are typically locked into a rather rigid search strategy. Modeling constraints independently from the search strategy also allows them to accomodate a variety of constraints, and therefore adapt the resulting pattern sets to the need of the user.

The key contribution of this paper is a method based on *integer linear programming* (ILP) that is (1) *exhaustive*, avoiding the loss of interesting solutions, (2) *declarative*, allowing us to make the most of provided constraints instead of being tied to a particular search strategy, (3) *optimal*, always returning the best solution according to an optimization criterion that satisfies the given constraints. Any measure that can be used as a constraint in an ILP model can also be chosen as an objective function to be optimized. This allows the user to prioritize particular aspects of the solution. Measures to be optimized can be combined, as long as they can be expressed as a linear term. This is once again an advantage over traditional mining, where a change would typically require the explicit redefinition of the search strategy. Our approach allows us to provide the first *practically useful* algorithm for addressing this problem setting. As an illustration, we experimentally address the tiling problem but our approach is broad enough to cover and leverage many pattern mining problems such as clustering [1,5], classification [14], or pattern summarization [21].

The rest of the paper is structured as follows. Section 2 recalls preliminaries. Section 3 describes our approach. Section 4 introduces several complex queries, and Section 6 shows the results of solving queries for the k-tiling problem on different data sets. We discuss related work in Section 5.

2 Preliminaries

2.1 Local patterns

Let $\mathcal{I}$ be a set of n distinct items, an itemset (or *pattern*) is a non-null subset of $\mathcal{I}$. The language of itemsets $\mathcal{L}$ corresponds to $2^{\mathcal{I}} \setminus \emptyset$. A *transactional dataset* $\mathcal{D}$ is a multi-set of m itemsets of $\mathcal{L}$, with each $t_i, 1 \leq i \leq m$ called a *transaction*. We assume the database $\mathcal{D}$ is represented as a binary matrix of size $n \times m$ with $\mathcal{D}_{ti} = 1 \leftrightarrow (t, i) \in \mathcal{D}$.

Let $\mathcal{D}$ a database, $\phi \in \mathcal{L}$ be a pattern, and $match : \mathcal{L} \times \mathcal{L} \mapsto \{true, false\}$ a matching operator. The cover of ϕ w.r.t $\mathcal{D}$, denoted by $cov(\phi, \mathcal{D})$, is the set of transactions in $\mathcal{D}$ that ϕ matches: $cov(\phi, \mathcal{D}) = \{t \in \mathcal{D} \mid match(\phi, t) = true\}$. The support of ϕ is the size of its cover: $sup(\phi, \mathcal{D}) = |cov(\phi, \mathcal{D})|$. The tile of a pattern ϕ contains all tuples that are covered by the pattern: $tile(\phi, \mathcal{D}) = \{(t, i) \mid t \in cover(\phi, D), i \in \phi\}$. These tuples form a tile or rectangle of 1's in the database D. The area of $tile(\phi, \mathcal{D})$ is equal to its cardinality: $area(\phi, \mathcal{D}) = |tile(\phi, \mathcal{D})| = |\phi| \,.\, sup(\phi, \mathcal{D})$.

A pattern ϕ is said to be *more general than* a pattern ψ ($\phi \preceq \psi$) (resp., ψ is *more specific* than ϕ) iff $\forall t \in \mathcal{L} : match(\psi, t) \Rightarrow match(\phi, t)$, i.e. if ψ matches any transaction t then ϕ matches it as well. A pattern ϕ is strictly more general than a pattern ψ ($\phi \prec \psi$), if $\phi \preceq \psi$ and $\neg(\psi \preceq \phi)$.

The local pattern mining problem consists of finding a theory $Th(\mathcal{L}, \mathcal{D}, q) = \{\phi \in \mathcal{L} \mid q(\phi, \mathcal{D}) \ is \ true\}$, where $q(\phi, \mathcal{D})$ a selection predicate that states the constraints under which the pattern ϕ is a solution w.r.t. the database $\mathcal{D}$. A common example is the minimum support constraint $sup(\phi, \mathcal{D}) \geq \theta$, which is satisfied by all patterns ϕ whose support in the database D exceeds a given minimal threshold θ. Combined with $\mathcal{L}$, this gives rise to the *frequent itemset mining* problem. An exact condensed representation of the frequent itemsets consists of the closed patterns [17]. A closed pattern is one whose specializations have a smaller cover than the pattern itself: $closed(\phi) \Leftrightarrow \forall \psi, \phi \preceq \psi : cov(\psi) \subset cov(\phi)$.

2.2 Pattern set mining

Pattern sets are simply sets of patterns. The task of pattern set mining entails discovering a set of patterns that satisfies a set of constraints involving not only individual patterns, as in the local pattern mining setting, but the whole set of patterns. Hereafter, we will denote by $\mathbf{L}$ the set of all the possible pattern sets that can be enumerated given a language $\mathcal{L}$, i.e., $\mathbf{L} = 2^{\mathcal{L}}$. The individual/local patterns occurring in a pattern set will be denoted using lower case characters such as $\phi, \ldots, \psi$ and for patterns sets, we will employ upper case characters such as $\Phi, \ldots, \Psi$.

More formally, the problem of pattern set mining can be formulated as the problem of computing the theory $\mathbf{Th}(\mathbf{L}, \mathcal{D}, p) = \{\Phi \in \mathbf{L} \mid p(\Phi, \mathcal{D}) \ is \ true\}$, where $p(\Phi, \mathcal{D})$ a selection predicate that states the constraints under which the pattern set Φ is a solution w.r.t. the database $\mathcal{D}$. 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 $f(\Phi)$, i.e. $\mathrm{argmax}_{\Phi \in \mathbf{Th}(\mathbf{L}, \mathcal{D}, p)} f(\Phi)$. In classical pattern set mining, this is achieved by dynamically increasing the threshold of the constraint involving f. When using the ILP framework (see Section 3), this can be achieved by using f as an objective function to guide the search.

2.3 Categories of Constraints

This section discusses several categories of constraints that can be specified at the level of the pattern set as a whole.

Coverage Constraints deal with defining and measuring how well a pattern set covers the data. Let $\Phi \in \mathbf{L}$ be a pattern set and $\mathcal{D}$ a database,

- *Pattern set cover*. The cover of Φ, denoted as $cov(\Phi, \mathcal{D})$, is the set of transactions in $\mathcal{D}$ that Φ covers: $cov(\Phi, \mathcal{D}) = \bigcup_{\phi \in \Phi} cov(\phi, \mathcal{D})$. With this definition, Φ is interpreted as the disjunction of the individual patterns ϕ it contains.
- *Support of pattern set*. The support of Φ, denoted as $sup(\Phi, \mathcal{D})$, is calculated in the same way as for individual patterns: $sup(\Phi, \mathcal{D}) = |cov(\Phi, \mathcal{D})|$.
- *Size of pattern set*. The size of Φ, denoted as $size(\Phi)$, is the number of patterns that Φ contains: $size(\Phi) = |\Phi|$.
- *Area of pattern set*. The area of a pattern set was studied in the context of large tile mining [6]. The area of Φ, denoted as $area(\Phi, \mathcal{D})$, is defined as the area of all the tiles of the individual patterns ϕ it contains: $area(\Phi, \mathcal{D}) = |\bigcup_{\phi \in \Phi} tile(\phi, \mathcal{D})|$.
- *Generality of pattern set*. A pattern set Φ is more general than a pattern set Ψ, denoted as $\Phi \preceq \Psi$, iff for all pattern $\psi \in \Psi$, there exists a pattern $\phi \in \Phi$ s.t. $\phi \preceq \psi$.

Discriminative Constraints. Given a database $\mathcal{D}$ organized into possibly overlapping subsets $\mathcal{D}_1, \ldots, \mathcal{D}_n \subseteq \mathcal{D}$, the discriminative constraints can be used to measure and optimize how well a pattern set discriminates between examples of subsets $\mathcal{D}_i$. Discriminative measures are typically defined by comparing the number of examples covered by the pattern set for a subset $\mathcal{D}_i$, to the total number of examples covered in $\mathcal{D}$.

– *Representativeness of a pattern set*. Representativeness indicates how characteristic the examples covered by the pattern set are for a subset $\mathcal{D}_i$. $rep(\Phi, \mathcal{D}_i, \mathcal{D}) = sup(\Phi, \mathcal{D}_i)/sup(\Phi, \mathcal{D})$.

– *Accuracy of a pattern set*. Let a dataset D partitioned into subsets $\mathcal{D}_1, \ldots, \mathcal{D}_n$, where each subset $\mathcal{D}_i$ contains transactions from class i. The accuracy of a pattern ϕ is defined as $acc(\phi) = \frac{\max_{\mathcal{D}_i \in \mathcal{D}} sup(\phi, D_i)}{sup(\phi, \mathcal{D})}$. The accuracy of an entire pattern set is harder to quantify. We can however *approximate* it as $acc(\Phi) = \frac{\sum_{\phi \in \Phi} acc(\phi) \times sup(\phi, \mathcal{D})}{\sum_{\phi \in \Phi} sup(\phi, \mathcal{D})}$

Redundancy Constraints can be used to constrain or minimize the redundancy between different patterns. One way to measure this redundancy is to count the number of transactions covered by multiple patterns in the pattern set Φ:

$red(\Phi, \mathcal{D}) = |\{t \in \mathcal{D} \mid \exists(\phi, \psi) \in \Phi, t \in ovlp(\phi, \psi, \mathcal{D})\}|$, where $ovlp(\phi, \psi, \mathcal{D}) = cov(\phi, \mathcal{D}) \cap cov(\psi, \mathcal{D})$ denotes the overlap between the two patterns ϕ, ψ [18].

3 Mining pattern sets using ILP

Throughout the remainder of this paper we employ the Integer Linear Programming (ILP) framework for representing and solving pattern set mining problems. ILP [15] is one of the most widely used methods for handling optimization problems, due to its rigorousness, flexibility and extensive modeling capability. This framework has been shown 1) to allow for the use of a wide range of constraints, 2) to offer a higher level of problem formalization and modeling, and 3) to work for conceptual clustering [16]. Moreover, modern ILP solvers are very efficient with improved search heuristics.

3.1 Resolution approach

Finding a good pattern set is often a hard task; many pattern set mining tasks and their optimization versions, such as the k-tiling [6] or the concept learning [11], are NP-hard. Hence, there are no straightforward algorithms for solving such tasks in general, giving rise to a wide ranges of approaches (Two-step vs one-step) and search strategies (Exact vs heuristic). In this paper, we adopt a **two-step** approach:
(i) A *local mining step* mines the set of local patterns $Th(\mathcal{L}, \mathcal{D}, q)$ that satisfy a set of constraints.
(ii) An *ILP mining step* post-processes these patterns with the **ILP exact** solving technique to obtain the best pattern set in $\mathbf{Th}(\mathbf{L}, \mathcal{D}, p)$ under the given constraints.

Our motivation for adopting a two-step approach is two-fold: First, there exist efficient miners [22] to find local patterns. Second, in the second step, the formulated ILP model (see Section 3.2) is very close to the well known partitioning (and covering) problems which are extensively studied within the integer programming community [10]. Modern ILP solvers, such as Cplex [8], are efficient on such problems. Our ILP model is detailed in the next section.

Constraint name	Notation	ILP formulation
$(C_{size}^{\theta,\lesseqgtr})$	$size(\Phi) \lesseqgtr \theta$	$\sum_{p\in\mathcal{P}} x_p \lesseqgtr \theta$
$(C_{cov}^{\theta,\lesseqgtr})$ $(C_{x,y})$	$sup(\Phi,\mathcal{D}) \lesseqgtr \theta$	$\sum_{t\in\mathcal{D}} y_t \lesseqgtr \theta$ $y_t \leq \sum_{p\in\mathcal{P}} a_{t,p}\cdot x_p \leq \lvert\mathcal{P}\rvert\cdot y_t, \forall t\in\mathcal{D}$
$(C_{area}^{\theta,\lesseqgtr})$ $(C_{x,q})$	$area(\Phi,\mathcal{D}) \lesseqgtr \theta$	$\sum_{i\in\mathcal{I},t\in\mathcal{D}} q_{t,i} \lesseqgtr \theta$ $q_{t,i} \leq \sum_{p\in\mathcal{P}} cq_{t,i}^{p} x_p \leq \lvert\mathcal{P}\rvert . q_{t,i}, \forall t\in\mathcal{D},\ \forall i\in\mathcal{I}$
$(C_{gen}^{\Phi_0})$	$\Phi \preceq \Phi_0$	$\sum_{\{p:\ \forall p\in\mathcal{P}\ \mid\ p\preceq\phi\}} x_p \geq 1, \quad \forall\phi\in\Phi_0$
$(C_{spe}^{\Phi_0})$	$\Phi_0 \preceq \Phi$	$\sum_{\{p:\ \forall p\in\mathcal{P}\ \mid\ \exists\phi\in\Phi_0,\ \phi\preceq p\}} x_p \geq \sum_{p\in\mathcal{P}} x_p$
$(C_{redd}^{\theta,\lesseqgtr})$ $(C_{x,u})$	$red(\Phi,\mathcal{D}) \lesseqgtr \theta$	$\sum_{t\in\mathcal{D}} u_t \lesseqgtr \theta$ $2u_t \leq \sum_{p\in\mathcal{P}} a_{t,p}x_p \leq y_t + \lvert\mathcal{P}\rvert . u_t, \forall t\in\mathcal{D}$
$(C_{rep}^{\theta,\lesseqgtr,D_i})$ $(C_{x,y'})$	$rep(\Phi,\mathcal{D}_i,\mathcal{D}) \lesseqgtr \theta$	$\sum_{t\in\mathcal{D}_i} y'_t \lesseqgtr \theta\times\sum_{t\in\mathcal{D}} y_t$ $y'_t \leq \sum_{p\in\mathcal{P}} a_{t,p}\cdot x_p \leq \lvert\mathcal{P}\rvert\cdot y'_t, \quad \forall t\in\mathcal{D}_i$
$(C_{avg}^{\theta,\lesseqgtr})$	$avg(sup(\Phi,\mathcal{D})) \lesseqgtr \theta$	$\sum_{p\in\mathcal{P}} f_p\cdot x_p - \theta\sum_{p\in\mathcal{P}} x_p \lesseqgtr 0$
$(C_{sum}^{\theta,\lesseqgtr})$	$sum(sup(\Phi,\mathcal{D})) \lesseqgtr \theta$	$\sum_{p\in\mathcal{P}} f_p\cdot x_p \lesseqgtr 0$

Table 1. ILP formulations of constraints discussed in Section 2.3.

3.2 ILP models

This section presents the ILP model of a pattern set, and the ILP formulations for constraints presented in Section 2.3.

Modeling a pattern set. Let $\mathcal{P}$ be the set of ℓ patterns. Hence, the pattern set mining problem can be modeled as an ILP using ℓ boolean variables $x_p (p \in \mathcal{P})$, where $(x_p = 1)$ iff the pattern p belongs to the unknown pattern set Φ that we are looking for.

Coverage Constraints. Let $\mathcal{D}$ be a database with m transactions. We introduce m boolean variables $y_t, (t \in \mathcal{D})$ such that $(y_t = 1)$ iff there exists at least one pattern $\phi \in \Phi$ such that pattern ϕ matches t. So, we have $sup(\Phi,\mathcal{D}) = \sum_{t\in\mathcal{D}} y_t$. With some given threshold θ, the coverage constraint $(C_{cov}^{\theta,\lesseqgtr})$ is defined on such boolean variables. Table 1 shows the formulation of the coverage constraints. Constraints $C_{x,y}$ establish the relationship between variables x and y, and state that each transaction t must belong to at most $|\mathcal{P}|$ patterns. Note that $(y_t = 0)$ iff there exists no pattern $\phi \in \Phi$ such that the pattern ϕ matches t.

Let $(a_{t,p})$ be an $m \times \ell$ binary matrix where $(a_{t,p} = 1)$ iff $match(p,t) = true$, i.e., the pattern p matches the transaction t. For the area constraint, we need to compute the number of ones in the binary matrix that are covered by the set of patterns. We can model this by introducing a temporary variable $q_{t,i}$ for every tuple (t,i), such that $(q_{t,i} = 1)$ iff there exists at least one tile (pattern) $\phi \in \Phi$ such that $t \in cover(\phi, D)$ and $i \in \phi$. Let $cq_{t,i}^{p}$ be a binary matrix associated to each pattern p where $(cq_{t,i}^{p} = 1)$ iff p covers both transaction t and item i. Constraints $(C_{x,q})$ establish the relationship between variables q and x.

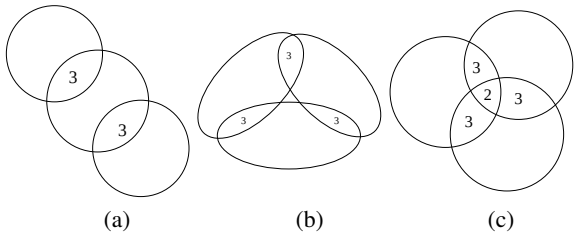

Fig. 1. Three patterns overlapping

Let Φ_0 a given pattern set. We model the generality constraint $(C_{gen}^{\Phi_0})$ as follows: $(\Phi \preceq \Phi_0)$ iff for any pattern $\phi \in \Phi_0$, there exists (at least) one pattern $p \in \Phi$ s.t. $p \preceq \phi$, i.e. $x_p = 1$. For the specialisation constraint $(C_{spe}^{\Phi_0})$, $(\Phi_0 \preceq \Phi)$ iff for any pattern $p \in \Phi$, there exists (at least) one pattern $\phi \in \Phi_0$ s.t. $\phi \preceq p$. "For any pattern $p \in \Phi$" is modeled by stating that the number of patterns $p \in \Phi$ verifying the property must be greater or equal to $size(\Phi) = \sum_{p \in \mathcal{P}} x_p$.

Handling aggregates. Table 1 shows how constraints involving aggregates (e.g. sum, avg, min, max) can be modeled using ILP. For example, the constraint $(C_{sum}^{\theta,\lessgtr})$ expresses that the sum of the supports over all patterns in Φ should be $\lessgtr$ than θ. It can be modeled using a linear constraint, where f_p is the support value of a local pattern p. Similarly, we can constraint the average taken over *all* patterns in Φ $(C_{avg}^{\theta,\lessgtr})$.

Redundancy constraints. To deal with redundancy, we need to know transactions that are multiply covered. Thus, we introduce boolean variables $(u_t), (t \in \mathcal{D})$ s.t. $(u_t = 1)$ iff transaction t is matched by at least 2 patterns. The total number of such transactions is $\sum_{t \in \mathcal{D}} u_t$. Table 1 gives the modelisation of the redundancy constraint $(C_{redd}^{\theta,\lessgtr})$, while constraints $(C_{x,u})$ establish the relationship between intermediate variables (u and y) and decision variables x.

Our definition of redundancy is similar to that proposed in [19] yet different from the (pairwise) redundancy proposed in [18]. The latter was adopted mainly due to its effectiveness for pruning in a level-wise mining algorithm. The differences between the two formalizations of redundancy are briefly sketched here.

Consider the three cases of overlapping patterns shown in Figure 1. The numbers in overlapping areas denote the number of transactions in the overlap. All three cases would be considered the same by a constraint measuring the maximal pairwise overlap between patterns, such as used in [18]. The global redundancy measure we employ evaluates to 6 for case (a), 9 for case (b), and 11 for case (c), capturing the actual situation much better. Notably, summing pairwise redundancies, another option proposed in that paper, will lead to a result of 15 for (c), overstating the redundancy. For this reason, we claim that our modelisation is more appropriate than the one chosen in [18].

Representativeness constraints. Let $\mathcal{D}_i \subseteq \mathcal{D}$ be a partial data set. We introduce $|\mathcal{D}_i|$ extra boolean variables $y'_t, (t \in \mathcal{D}_i)$ such that $(y'_t = 1)$ iff there exists at least one pattern $\phi \in \Phi$ such that pattern ϕ matches t. The modelisation of the representativeness constraint $(C_{rep}^{\theta,\lessgtr,D_i})$ is shown in Table 1. Constraints $(C_{x,y'})$ establish the relationship between variables (y'_t) and (x_p).

4 Queries and how to model them

This section provides three examples of complex queries and shows how to model them as a combination of constraints presented in Section 2.3.

As a first query (Q_1), we show the quintessential pattern set mining task: given the result of a local pattern mining operation, we aim to find a (relatively) small subset of patterns that is representative of the entire result. To this end, we want to select patterns that have very little overlap and together cover as much of the data as possible. There are two ways of modeling this query as an ILP problem :

$$\begin{cases} \texttt{Maximize } \sum_{t\in\mathcal{D}} y_t \\ (C^{\theta,\leq}_{redd}) \\ (C_{x,y,u}) \\ x_p \in \{0,1\}, \forall p \in \mathcal{P} \\ y_t \in \{0,1\}, u_t \in \{0,1\}, \forall t \in \mathcal{D} \end{cases} \quad (1) \qquad \begin{cases} \texttt{Minimize } \sum_{t\in\mathcal{D}} u_t \\ (C^{\theta,\geq}_{cov}) \\ (C_{x,y,u}) \\ x_p \in \{0,1\}, \forall p \in \mathcal{P} \\ y_t \in \{0,1\}, u_t \in \{0,1\}, \forall t \in \mathcal{D} \end{cases} \quad (2)$$

(1) *maximize support* ($z = \sum_{t\in\mathcal{D}} y_t$) subject to maximum redundancy constraints;
(2) *minimize redundancy* ($z = \sum_{t\in\mathcal{D}} u_t$) subject to minimum support constraints. Constraints $(C_{x,y})$ and $(C_{x,u})$ governing the new variables u and y can be merged and will be denoted as the following linear constraints $(C_{x,y,u})$: $y_t + u_t \leq \sum_{p\in\mathcal{P}} a_{t,p}x_p \leq y_t + |\mathcal{P}| \,.\, u_t, y_t \geq u_t, \forall t \in \mathcal{D}$

Our second query (Q_2) is a refinement of the first one, by imposing a generality constraint. For instance, we aim to summarize a set of subgraphs mined from molecular data, so that a non-data miner, e.g. a chemist, has only a small set of fragments to evaluate. In that case, the practitioner might already have an idea what fragments she would like to see, and wants to see the rest fleshed out. This can be achieved by requiring that a pattern set include a particular pattern (or syntactically related patterns), i.e. that is *more general* than another one. In the ILP case as well, this just requires a generality constraint to be added.

Our last query (Q_3) concerns the k-*tiling*. The task consists of finding k tiles maximizing the area ($z = \sum_{t\in\mathcal{D},i\in\mathcal{I}} q_{t,i}$). Equation (3) depicts our first ILP model `M1`. The number of tiles k can also be defined as a variable whose value will be determined by the ILP solver. This can be done by specifying a lower bound k_{min} and/or an upper bound k_{max} on the value of k. Note that tiles can be overlapping, but every tuple (t, i) covered is only counted once. This encoding requires $(n \times m + 2)$ constraints and $((n \times m) + \ell + 1)$ variables. This constitutes a major limitation when it comes to handling very large databases. Thus, we propose a second ILP model `M2` that approximates the k-tiling by summing the areas v_p $(p \in \mathcal{P})$ of the individual tiles it contains. In this case, each tuple (t, i) covered may be counted more than once.

$$\begin{cases} \texttt{Maximize } \sum_{t\in\mathcal{D},i\in\mathcal{I}} q_{t,i} \\ (C_{x,q}) \\ k = \sum_{p\in\mathcal{P}} x_p \\ k_{min} \leq k \leq k_{max} \\ k \in \mathbb{N}, \\ x_p \in \{0,1\}, \\ \quad p \in \mathcal{P} \\ q_{i,t} \in \{0,1\}, \\ \quad i \in \mathcal{I} \wedge t \in \mathcal{D} \end{cases} \quad (3) \qquad \begin{cases} \texttt{Maximize } \sum_{p\in\mathcal{P}} v_p \,.\, x_p \\ (1)\ y_t \leq \sum_{p\in\mathcal{P}} a_{t,p} \,.\, x_p \leq \delta_o \,.\, y_t, \quad \forall t \in \mathcal{D} \\ (2)\ \sum_{t\in\mathcal{D}} y_t \geq \theta_t \\ (3)\ z_i \leq \sum_{p\in\mathcal{P}} w_{i,p} \,.\, x_p \leq \gamma_o \,.\, z_i, \quad \forall i \in \mathcal{I} \\ (4)\ \sum_{i\in\mathcal{I}} z_i \geq \theta_i \\ k = \sum_{p\in\mathcal{P}} x_p \\ k_{min} \leq k \leq k_{max} \\ k \in \mathbb{N}, \\ x_c \in \{0,1\}, \quad c \in \mathcal{P} \\ y_t \in \{0,1\}, \quad t \in \mathcal{D} \\ z_i \in \{0,1\}, \quad i \in \mathcal{I} \end{cases} \quad (4)$$

Equation (4) depicts our second ILP model M2. It consists of finding a set of tiles covering both the set of transactions and the set of items, with small overlaps on transactions and on items. In this way, we allow to control the redundancy on tuples (t, i) that are multiply counted in the tiling. Constraint (2) states that at least θ_t transactions must be covered, while Constraint (1) states that each transaction t cannot occur in more than δ_o closed patterns. Let $w_{i,p}$ be an $n \times \ell$ binary matrix where $(w_{i,p} = 1)$ iff the item i belongs to the tile or pattern p. Constraint (4) states that at least θ_i items must be covered, while Constraint (3) states that each item i cannot occur in more than γ_o tiles. In the experimental section, due to the space limitation, we focus on questions related to the k-tiling problem.

5 Related work

Pattern set mining is an important part of a number of data mining tasks such as classification [14], clustering [1,5], pattern summarization [21], or database tiling [6]. Due to the highly combinatorial nature of the problem, most methods proceed in two phases. First, an exhaustive algorithm generates the whole collection of local patterns satisfying given constraints and the second phase produces pattern sets by selecting smaller subsets of relevant patterns from the whole collection of local patterns, often by using heuristics to manage computational complexity. Unlike these works, our method does not use heuristics to provide a pattern set but relies on a solid formalization in the ILP paradigm, while the search is guided by the optimization of an objective function.

There are very few attempts on searching pattern sets according to complete approaches. The original idea of pattern set mining was proposed in [18]. The authors formally introduce a variety of constraints at the level of the pattern sets. Unfortunately, the pruning techniques are weak and the algorithm remains limited to small collections of patterns. Specific settings have been proposed by [13] (fixing the size of pattern sets and relying on the anti-monotonicity of a particular quality function) or [3] (using a dedicated global constraint on the attributes). The approach in [6] is heuristic but does *not* use post-processing, instead iteratively mining tiles, taking already found ones into account.

The constraint programming framework was investigated to accomplish the pattern set mining task by modeling pattern sets with constraints [7,12]. However, these methods require to fix the number of local patterns included in a pattern set, a strong limitation in practice, and tend to have scaling problems. Recent contributions also employ more specialized systems such as satisfiability solvers [9] and integer linear programming techniques [1,16]. These methods address particular problem settings whereas we propose a declarative and exhaustive method based on ILP returning the best solution according to an optimization criterion, and which is able to handle a wide variety of constraints and therefore various different pattern set mining tasks.

6 Experiments

For a better understanding of the suitability of the ILP approach, we focus our experiments on one prototypical task for pattern set mining: the k-tiling problem (NP-Hard).

The experimental evaluation is designed to address the following questions:
1. How do the running times of our approach (`ILP`) compare to the only existing CP approach, proposed by Guns et al. (`KPatternSet`), for the k-tiling problem?
2. Given the space requirements of the ILP model `M1`, how do the obtained k-tilings compare qualitatively with those resulting from the (approximating) ILP model `M2`?
3. In light of the exact nature of our approach, how do the resulting k-tilings compare qualitatively with those resulting from (`k-LTM`) [6].[3]
4. How do the k-tilings with overlapping tiles compare qualitatively with those having non-overlapping tiles ?

Experimental protocol. Experiments were carried out on the same datasets which were used in [7] and available from the UCI repository. Table 2 shows the characteristics of these datasets. All experiments were conducted on AMD Opteron 6174 with 2.60 GHz of CPU and 256 GB of RAM.[4] We used closed patterns to represent tiles since they cover a larger area than their non-closed counterparts. We used `LCM` to extract the set of all closed patterns and `CPLEX` $v.12.4$ to solve the different ILP models. For all methods, a time limit of 24 hours has been used. As `M2` requires setting the parameters for the coverage and non-overlap relations, we propose the following three settings :
- `M2a` with settings allowing similar amounts of coverage and overlap as `k-LTM`;
- `M2b` with settings allowing the coverage of all transactions ($\theta_i = m$) with the maximum overlap ($\delta_0 = |\mathcal{P}|$) and the coverage of at least one item ($\theta_i = 1$) without any overlap for items ($\gamma_0 = 1$);
- `M2c` with settings allowing the coverage of all transactions ($\theta_i = m$) without any overlap ($\delta_0 = 1$) and the coverage of at least one item ($\theta_i = 1$) with the maximum overlap for items ($\gamma_0 = |\mathcal{P}|$).

To assess the quality of a k-tiling Φ, we define the *recall* of Φ, measured by the fraction of all ones in the binary matrix $\mathcal{D}$ belonging to $area(\Phi, \mathcal{D})$, which should be as large as possible: $recall(\Phi, \mathcal{D}) = \frac{\sum_{(t,i) \in area(\Phi,\mathcal{D})} \mathcal{D}_{ti}}{\sum_t \sum_i \mathcal{D}_{ti}}$.

(a) Comparing `ILP-M1` with `KPatternSet`. Tab. 2 compares the performance compares of `ILP-M1` and `KPatternSet`(in terms of CPU-times) for various values of k on different datasets. We also report the corresponding value of recall. The CPU-times of `ILP-M1` include those for the preprocessing step. `ILP-M1` clearly outperforms `KPatternSet` on all datasets: `KPatternSet`goes over the timeout for $k \geq 4$. For the value of k for which an optimal k-tiling can be found, `ILP-M1` is up to several orders of magnitude faster than `KPatternSet`.

(b) Comparing `M1` with `M2a` and `M2b`. `ILP-M1` finds the optimal solution on most of the datasets, but `ILP-M2a` remains relatively close in terms of recall, particularly for ($k \leq 5$). In addition, `ILP-M2a` is much faster, particularly on Lymph and Vote (speed-up of up to 660). The main limitation of `M1` remains its space requirement. For the three most difficult datasets – Mushroom, Hepatitis and Anneal – `ILP-M1` fails to find a solution. Comparing `ILP-M2a` with `ILP-M2b`, the latter shows clearly higher recall

[3] We use the implementation available at `https://people.mmci.uni-saarland.de/~jilles/prj/tiling/`.
[4] The `k-LTM` implementation is Windows-only, and run times therefore only roughly indicate its behavior.

$\mathcal{D}$	Items	Trans.	k	Recall					Time (s)				
				(1)	(2)	(3)	(4)	(5)	(1)	(2)	(3)	(4)	(5)
Zoo-1	34	101	3	0.48	0.38	0.38	0.48	0.46	8.99	5.87	1.57	2,418	
			4	0.57	0.47	0.48	TO	0.56	14.38	4.53	3.67	TO	
			5	0.63	0.54	0.52	TO	0.62	25.11	4.49	2.99	TO	
			6	0.68	0.55	0.55	TO	0.67	34.96	10.82	2.98	TO	
			7	0.73	0.57	0.58	TO	0.71	22.85	5.81	3.49	TO	
			8	0.78	0.68	0.61	TO	0.75	25.56	6.96	3.16	TO	
			9	0.82	0.69	0.64	TO	0.79	35.07	8.04	3.59	TO	
			10	0.85	0.69	0.66	TO	0.82	26.21	6.91	3.23	TO	0.99
Lymph	66	148	3	0.43	0.42	0.35	TO	0.42	299.91	50.92	152.54	TO	
			4	0.48	0.38	0.39	TO	0.47	954.74	143.84	152.13	TO	
			5	0.52	0.44	0.42	TO	0.51	1,629	97.88	152.72	TO	
			6	0.55	0.37	0.45	TO	0.54	4,261	175.01	152.52	TO	
			7	0.58	0.40	0.46	TO	0.57	6,027	155.19	153.15	TO	
			8	0.61	0.40	0.48	TO	0.59	5,197	47.68	146.13	TO	
			9	0.63	0.40	0.49	TO	0.61	7,313	45.51	148.06	TO	
			10	0.65	0.44	0.50	TO	0.63	28,779	43.56	148.51	TO	1321
Primary-tumor	29	336	3	0.46	0.40	0.36	0.46	0.46	405.91	33.25	82.26	41,496	
			4	0.54	0.48	0.47	TO	0.53	680.64	74.99	88.60	TO	
			5	0.59	0.47	0.55	TO	0.57	3,807	43.93	68.65	TO	
			6	0.64	0.44	0.60	TO	0.61	3,863	92.28	58.76	TO	
			7	0.68	0.46	0.64	TO	0.65	6,407	154.82	61.58	TO	
			8	0.71	0.50	0.67	TO	0.68	86,429	41.75	53.93	TO	
			9	0.74	0.54	0.70	TO	0.71	86,425	65.50	65.13	TO	
			10	0.77	0.52	0.73	TO	0.73	86,423	43.70	58.23	TO	4.9
Soybean	47	630	3	0.44	0.42	0.33	0.44	0.43	127.94	15.31	8.35	32,485	
			4	0.50	0.47	0.37	TO	0.49	147.11	24.38	6.96	TO	
			5	0.55	0.49	0.41	TO	0.53	141.70	22.10	6.43	TO	
			6	0.59	0.47	0.43	TO	0.58	144.27	19.25	6.08	TO	
			7	0.62	0.47	0.45	TO	0.60	157.41	14.11	5.90	TO	
			8	0.65	0.45	0.47	TO	0.62	358.05	16.30	9.65	TO	
			9	0.67	0.46	0.48	TO	0.64	342.79	14.65	8.83	TO	
			10	0.69	0.48	0.49	TO	0.66	734.20	9.42	6.28	TO	7

$\mathcal{D}$	Items	Trans.	k	Recall					Time (s)				
				(1)	(2)	(3)	(4)	(5)	(1)	(2)	(3)	(4)	(5)
Tic-Tac-Toe	24	958	3	0.15	0.15	0.11	TO	0.15	24.49	8.93	5.98	TO	
			4	0.20	0.20	0.16	TO	0.20	23.67	6.11	5.05	TO	
			5	0.25	0.25	0.21	TO	0.25	25.81	5.43	5.82	TO	
			6	0.29	0.29	0.27	TO	0.29	24.45	5.92	4.91	TO	
			7	0.33	0.33	0.33	TO	0.33	43.17	5.53	2.70	TO	
			8	0.38	0.38	0.38	TO	0.38	44.76	5.44	3.82	TO	
			9	0.42	0.42	0.42	TO	0.42	42.76	4.87	3.50	TO	
			10	0.46	0.46	0.46	TO	0.46	41.98	5.16	3.35	TO	0.1
Vote	45	435	3	0.32	0.30	0.06	TO	0.31	3,020	81.75	34.36	TO	
			4	0.37	0.35	0.19	TO	0.36	1,947	84.99	46.09	TO	
			5	0.42	0.38	0.33	TO	0.40	4,269	732.10	49.42	TO	
			6	0.46	0.39	0.40	TO	0.44	3,651	83.48	43.23	TO	
			7	0.50	0.38	0.45	TO	0.47	13,831	282.56	39.13	TO	
			8	0.53	0.37	0.49	TO	0.51	8,671	819.33	33.03	TO	
			9	0.56	0.42	0.53	TO	0.54	15,152	881.76	36.10	TO	
			10	0.60	0.43	0.57	TO	0.57	13,974	701.37	30.53	TO	3.3
Mushroom	112	8124	3	TO	0.34	–	TO	0.34	TO	431.82	–	TO	
			4	TO	0.40	–	TO	0.40	TO	667.61	–	TO	
			5	TO	0.46	–	TO	0.46	TO	780.05	–	TO	
			6	TO	0.48	–	TO	0.50	TO	572.54	–	TO	
			7	TO	0.49	–	TO	0.53	TO	502.91	–	TO	
			8	TO	0.44	–	TO	0.56	TO	1,302	–	TO	
			9	TO	0.48	–	TO	0.58	TO	763.20	–	TO	
			10	TO	0.51	–	TO	0.60	TO	479.45	–	TO	666.4
hepatitis	66	137	3	TO	0.27	0.25	TO	0.28	TO	1,261	604.16	TO	
			4	TO	0.33	0.33	TO	0.34	TO	910.38	1,007.34	TO	
			5	TO	0.37	0.38	TO	0.39	TO	1,310	725.38	TO	
			6	TO	0.42	0.42	TO	0.44	TO	887.81	781.88	TO	
			7	TO	0.43	0.46	TO	0.48	TO	1,327	616.90	TO	
			8	TO	0.45	0.49	TO	0.52	TO	2,372	736.19	TO	
			9	TO	0.45	0.52	TO	0.55	TO	3,046	625.16	TO	
			10	TO	0.45	0.54	TO	0.57	TO	16,295	656.40	TO	6103

Table 2. Comparing the different approaches. (TO: TimeOut; – : no solution ; (1): ILP−M1; (2): ILP−M2a; (3): ILP−M2b; (4): KPatternSet; (5): k−LTM).

for Hepatitis, Primary Tumor and Vote. This is because ILP−M2b allows overlapping tiles with high redundancy (see Tab. 3). In addition ILP−M2b is faster than ILP−M2a on 51 instances (out of 64) with a speed-up between 1 and 4 for 27, and between 7 to 25 for 9 instances. Note that on Mushroom dataset, no k-tiling exists with M2b. These results show that ILP−M2a and ILP−M2b achieve good recall compared to ILP−M1 with less space requirement, hence the interestingness of using ILP−M2b as a faster alternative for approximating the optimal k-tiling.

(c) Comparing ILP with k−LTM. k−LTM differs from our approach in three points: 1) using a heuristic is faster but may lead to suboptimal solutions, 2) mining iteratively, k−LTM can take information about already found tiles into account, and 3) a k−LTM $k+1$-tiling will always be a superset of a k-tiling – ILP can find different solutions. As Tab. 2 shows, ILP−M1 always achieves better recall than k−LTM, yet requires more time to find the optimum, (k−LTM running times are shown in the last column). For the most difficult dataset Anneal, neither method find a solution. Comparing ILP−M2b with k−LTM, k−LTM has a slight advantage (three data sets). While complete search beats iterative mining, it *does* help gaining an advantage over heuristic post-processing.

(d) Comparing M2c with M2b and k−LTM. In our last experiment, we mine k tiles without any overlap (i.e. M2c) and compare them to those resulting from M2b and k−LTM. Tab. 3 shows four distinct cases of recall and redundancy for the three approaches. Col. 4 reports the redundancy of the k-tiling measured by $red(\Phi, \mathcal{D})/sup(\Phi, \mathcal{D})$. Col. 5 (resp 6) denotes the percentage of redundant (resp. covered) items. Generally,

$\mathcal{D}$	k	Recall		Redundant Trans.			Redundant Items			Coverage Items		
		(3)	(3')	(3)	(3')	(5)	(3)	(3')	(5)	(3)	(3')	(5)
Mushroom	3	-	0.23	-	0.00	0.55	-	0.50	0.31	-	0.08	0.16
	4	-	0.31	-	0.00	0.64	-	0.32	0.28	-	0.16	0.18
	5	-	0.36	-	0.00	0.81	-	0.42	0.29	-	0.22	0.24
	6	-	0.42	-	0.00	0.91	-	0.44	0.41	-	0.21	0.25
	7	-	0.46	-	0.00	0.92	-	0.59	0.55	-	0.24	0.25
	8	-	0.51	-	0.00	0.93	-	0.66	0.53	-	0.24	0.26
	9	-	0.54	-	0.00	0.93	-	0.69	0.51	-	0.24	0.27
	10	-	0.56	-	0.00	0.93	-	0.53	0.50	-	0.32	0.28
Soybean	3	0.33	0.28	0.85	0.00	0.59	0.00	0.30	0.41	0.16	0.20	0.25
	4	0.37	0.34	0.90	0.00	0.64	0.00	0.54	0.37	0.20	0.26	0.34
	5	0.41	0.40	0.93	0.00	0.74	0.00	0.56	0.47	0.22	0.36	0.36
	6	0.43	0.44	0.96	0.00	0.85	0.00	0.57	0.44	0.24	0.46	0.38
	7	0.45	0.48	0.98	0.00	0.85	0.00	0.52	0.52	0.26	0.42	0.40
	8	0.47	0.52	1.00	0.00	0.90	0.00	0.50	0.57	0.26	0.52	0.40
	9	0.48	0.54	1.00	0.00	0.93	0.00	0.62	0.55	0.28	0.52	0.42
	10	0.49	0.66	1.00	0.00	0.95	0.00	0.69	0.52	0.28	0.52	0.44

$\mathcal{D}$	k	Recall		Redundant Trans.			Redundant Items			Coverage Items		
		(3)	(3')	(3)	(3')	(5)	(3)	(3')	(5)	(3)	(3')	(5)
Primary-tumor	3	0.36	0.15	0.87	0.00	0.60	0.00	0.33	0.25	0.23	0.19	0.41
	4	0.47	0.22	0.90	0.00	0.71	0.00	0.38	0.38	0.35	0.26	0.44
	5	0.55	0.28	0.94	0.00	0.82	0.00	0.33	0.42	0.42	0.39	0.48
	6	0.60	0.33	0.96	0.00	0.84	0.00	0.50	0.53	0.48	0.39	0.51
	7	0.64	0.38	0.98	0.00	0.89	0.00	0.44	0.62	0.52	0.52	0.55
	8	0.67	0.42	1.00	0.00	0.91	0.00	0.42	0.75	0.52	0.61	0.55
	9	0.70	0.45	1.00	0.00	0.92	0.00	0.45	0.87	0.52	0.65	0.55
	10	0.73	0.48	1.00	0.00	0.95	0.00	0.45	0.77	0.58	0.71	0.62
Anneal	3	-	0.36	-	0.00	TO	-	0.47	TO	-	0.34	TO
	4	-	0.46	-	0.00	TO	-	0.64	TO	-	0.35	TO
	5	-	0.52	-	0.00	TO	-	0.74	TO	-	0.37	TO
	6	-	0.57	-	0.00	TO	-	0.67	TO	-	0.46	TO
	7	-	0.59	-	0.00	TO	-	0.84	TO	-	0.47	TO
	8	-	0.62	-	0.00	TO	-	0.71	TO	-	0.56	TO
	9	-	0.64	-	0.00	TO	-	0.78	TO	-	0.59	TO
	10	-	0.65	-	0.00	TO	-	0.82	TO	-	0.59	TO

Table 3. Qualitative comparison. ((3): `ILP-M2b`; (3'): `ILP-M2c`; (5): `k-LTM`).

`ILP-M2b` and `k-LTM` achieve higher recall than `ILP-M2c`. This is not surprising as the tilings found by `ILP-M2b` consist of large, transaction-overlapping tiles, contrary to those of `ILP-M2c` (see Col. 4). `ILP-M2c` makes up for this by covering more *items* than the other approaches, and mining item-overlapping tiles. This tuning of output characteristics is a strength of the declarative approach.

7 Conclusion

Pattern set mining has become an indispensable data mining task to control the overly large result sets of local pattern mining operations. In this work, we have for the first time presented a practically useful approach that retains the richness of the constraint language of the original pattern set mining approach [18]. Our method is declarative, based on the techniques developed in ILP, allowing to choose particular (combinations of) pattern set measures as objective functions to be optimized. This permits the user to prioritize particular aspects of a pattern set, while constraining others. Existing ILP solver guarantee to return the *best* pattern set, according to the given optimization criterion, that satisfies a user-defined set of constraints. Experiments have illustrated and shown the efficiency of our approach through the example of the tiling problem but our approach is broad enough to cover and leverage many pattern mining problems such as clustering, classification, or pattern summarization. The flexibility of our approach is clearly a major step towards developing the interactive data mining systems that are requested in data science. Having defined this framework, further work will consist of properly specifying the models corresponding to different data mining tasks. It will be necessary to formulate new constraints and models, and fine-tune them to achieve best results. We also plan to exploit column generation techniques to enhance the scalability of our solving step.

References

1. B. Babaki, T. Guns, and S. Nijssen. Constrained clustering using column generation. In *CPAIOR 2014, Cork, Ireland, May 19-23, 2014.*, pages 438–454, 2014.

2. B. Bringmann and A. Zimmermann. One in a million: picking the right patterns. *Knowledge and Information Systems*, 18(1):61–81, 2009.
3. L. Cagliero, S. Chiusano, P. Garza, and G. Bruno. Pattern set mining with schema-based constraint. *Knowl.-Based Syst.*, 84:224–238, 2015.
4. H. Cheng, X. Yan, J. Han, and C.-W. Hsu. Discriminative frequent pattern analysis for effective classification. In *ICDE 2007, Istanbul, Turkey, April 15*, pages 716–725, 2007.
5. M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD'96, Portland*, pages 226–231, 1996.
6. F. Geerts, B. Goethals, and T. Mielikäinen. Tiling databases. In *DS'04, Padova, Italy, October 2-5, 2004, Proceedings*, pages 278–289, 2004.
7. T. Guns, S. Nijssen, and L. De Raedt. k-pattern set mining under constraints. *IEEE Trans. Knowl. Data Eng.*, 25(2):402–418, 2013.
8. IBM/ILOG, Inc. ILOG CPLEX: High-performance software for mathematical programming and optimization, 2016.
9. S. Jabbour, L. Sais, and Y. Salhi. The top-k frequent closed itemset mining using top-k SAT problem. In *ECML PKDD'13, Prague, Czech Republic*, pages 403–418, 2013.
10. M. Jünger, T. M. Liebling, D. Naddef, G. L. Nemhauser, W. R. Pulleyblank, G. Reinelt, G. Rinaldi, and L. A. Wolsey, editors. *50 Years of Integer Programming 1958-2008 - From the Early Years to the State-of-the-Art*. Springer, 2010.
11. M.J. Kearns and U.V Vazirani. An introduction to computational learning theory, 1994.
12. M. Khiari, P. Boizumault, and B. Crémilleux. Constraint programming for mining n-ary patterns. In *CP'10, St. Andrews, Scotland, UK*, pages 552–567, 2010.
13. A. J. Knobbe and E. K. Y. Ho. Maximally informative k-itemsets and their efficient discovery. In *ACM SIGKDD'06, Philadelphia, PA, USA*, pages 237–244, 2006.
14. B. Liu, W. Hsu, and Y. Ma. Integrating classification and association rules mining. In *proceedings of Fourth International Conference on Knowledge Discovery & Data Mining (KDD'98)*, pages 80–86, New York, 1998. AAAI Press.
15. Andrzej J. O. Integer and combinatorial optimization. *International Journal of Adaptive Control and Signal Processing*, 4(4):333–334, 1990.
16. A. Ouali, S. Loudni, Y. Lebbah, P. Boizumault, A. Zimmermann, and L. Loukil. Efficiently finding conceptual clustering models with integer linear programming. In *IJCAI'16, New York, NY, USA, 9-15 July 2016*, pages 647–654, 2016.
17. N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Discovering frequent closed itemsets for association rules. In *ICDT'99, Jerusalem, Israel*, pages 398–416, 1999.
18. L. De Raedt and A. Zimmermann. Constraint-based pattern set mining. In *SIAM'07, April 26-28, 2007, Minneapolis, Minnesota, USA*, pages 237–248, 2007.
19. Y. Shima, K. Hirata, and M. Harao. Extraction of frequent few-overlapped monotone DNF formulas with depth-first pruning. In *PAKDD'05, Hanoi, Vietnam*, pages 50–60, 2005.
20. J. Vreeken, M. v. Leeuwen, and A. Siebes. Krimp: mining itemsets that compress. *Data Min. Knowl. Discov.*, 23(1):169–214, 2011.
21. D. Xin, H. Cheng, X. Yan, and J. Han. Extracting redundancy-aware top-k patterns. In *ACM SIGKDD'06, Philadelphia, PA, USA, August 20-23, 2006*, pages 444–453, 2006.
22. W. Xindong and K. Vipin. *The Top Ten Algorithms in Data Mining*. Chapman & Hall/CRC, 1st edition, 2009.