Modeling Pattern Set Mining using Boolean Circuits

John O.R. Aoga[0000−0002−7213−146X]⋆, Siegfried Nijssen[0000−0003−2678−1266], and Pierre Schaus[0000−0002−3153−8941]

ICTEAM/INGI, UCLouvain, Belgium
{john.aoga,siegfried.nijssen,pierre.schaus}@uclouvain.be

Abstract. Researchers in machine learning and data mining are increasingly getting used to modeling machine learning and data mining problems as parameter learning problems over network structures. However, this is not yet the case for several pattern set mining problems, such as concept learning, rule list learning, conceptual clustering, and Boolean matrix factorization. In this paper, we propose a new modeling language that allows modeling these problems. The key idea in this modeling language is that pattern set mining problems are modeled as discrete parameter learning problems over *Boolean circuits*. To solve the resulting optimisation problems, we show that standard optimization techniques from the constraint programming literature can be used, including mixed integer programming solvers and a local search algorithm. Our experiments on various standard machine learning datasets demonstrate that this approach, despite its genericity, permits learning high quality models.

1 Introduction

A revolution is taking place in artificial intelligence, driven to a significant degree by *deep learning* toolkits for learning neural networks [26]. As a result, researchers and practitioners in machine learning and data mining are increasingly getting used to modeling and solving problems using the modeling languages offered in these toolkits. The key idea underlying these languages is that machine learning amounts to learning the parameters of a network structure that transforms inputs into predicted outputs.

However powerful these toolkits may be, they rely on a key underlying assumption: the functions applied to the inputs are continuous and differentiable in the parameters. This enables the use of gradient descent to identify values for the parameters.

Some problems in data mining and machine learning are however not continuous in nature. Good examples of such problems can be found in pattern *set* mining. Examples of pattern set mining problems include learning rule-based classifiers, conceptual clustering, and Boolean matrix factorization. In each of these problems, the task is not to find values for a set of continuous parameters, but to identify some discrete *patterns*.

This raises the question as to whether a generic framework exists that allows formalizing and solving these types of problems using a modeling language similar to the language that deep learning researchers are familiar with. This is the challenge that we address in this paper.

⋆ The first author is supported by the FRIA-FNRS (Fonds pour la Formation à la Recherche dans l'Industrie et dans l'Agriculture, Belgium).

Generic modeling languages for pattern mining already exist in the literature. Well-studied is the *constraint programming for pattern mining* (CP4PM) framework [13,25]. However, this framework has two weaknesses. First, the modeling language is different from the network-based modeling language used in deep learning toolkits: pattern mining problems are formalized by modeling these problems using Boolean variables and constraints on these variables. Second, most of the studies on CP4PM focus on *pattern mining* instead of pattern *set* mining [14]. There is an important difference between pattern mining and pattern *set* mining. In pattern mining, one is interested in finding *all* patterns occurring in a dataset that satisfy a given set of constraints. The most well-known example of such a constraint is the requirement that a pattern is *frequent*. In pattern *set* mining, however, we are not interested in finding all patterns, but we are interested in finding a small set of patterns that together solve a well-defined data mining problem well, such as a classification task. Given the large number of frequent patterns that can typically be found in many datasets, in recent years, it has been argued that pattern set mining is the more relevant problem.

Only a limited number of studies have explored the extension of CP4PM to pattern set mining problems [14,12]. In these studies, a modeling language was proposed in which pattern set mining problems are formalized as constraint optimization problems. A solution strategy was proposed based on the use of a constraint programming solver. Unfortunately, this approach could, in practice, only be applied to relatively small datasets. Moreover, the modeling language proposed was rather different than the one currently used in machine learning toolkits.

We aim to address these weaknesses in this work. In this paper, we propose a new framework for **modeling pattern set mining problems**. Compared to existing frameworks, this framework has the following advantages:

- the modeling language that we propose is inspired by that of frameworks for deep learning: in our language, we represent learning problems as parameter learning problems on networks; however, instead of continuous functions, in the internal nodes of our networks we use Boolean operators, effectively turning the networks into *Boolean circuits*;
- the framework supports the generic use of a number of different solvers. In particular, in this work we will show how Mixed Integer Programming (MIP) and local search algorithms can be used to solve pattern set mining problems in a generic manner;
- as a consequence of the support of different solvers, the framework allows for finding larger pattern sets on larger datasets.

We introduce a domain specific language to ease the description of the network, its parameters, and the loss function to be computed on the data. Our experiments demonstrate the practicability and flexibility on standard benchmarks.

2 Related Work

A number of different modeling languages for pattern mining problems have already been proposed in the literature.

An important class of methods is based on the use of constraints to model pattern mining problems. The key idea in these approaches is to model a data mining problem as a *Constraint satisfaction problem (CSP)* and use a SAT-, CP- or MIP-solver for solving it. Several data mining tasks were studied, but most results were obtained for itemset mining [6,13,18,25] and sequence mining [3,16,21]. However, these approaches are *solver- and task-dependent*, and do not solve pattern *set* mining problems.

Most related to this work is the work of De Raedt [8] and Guns et al. [14] on modeling pattern set mining problems as constraint satisfaction or optimization problems. Contrary to these earlier works the framework that we propose is solver independent and uses a modeling language familiar to machine learning researchers.

A modeling language for pattern mining that is solver independent is the *MiningZinc* [11] language. However, it does not address pattern set mining problems either.

The first modeling languages that were proposed in the data mining literature are those that use an SQL-like notation [19]. Also, these languages did not study pattern set mining problems and do not use a notation based on networks.

3 Pattern Set Mining Problems

In this section we will introduce the pattern set mining problems that are the focus of this work. We limit our attention to Boolean data that may or may not be supervised in nature. Let $\mathcal{I} = \{1, \ldots, m\}$ be the sets of items (features) and $\mathcal{T} = \{1, \ldots, n\}$ be the set of transaction (observation) identifiers. An *unsupervised* database can equivalently be seen as a set $\mathcal{D} = \{(t, T_t) \mid t \in \mathcal{T}, T_t \subseteq \mathcal{I}\}$ or as a matrix such that $\mathcal{D}_{ti} \in \{0, 1\}$ where $t \in \mathcal{T}$ and $i \in \mathcal{I}$. In a *supervised* database, we associate with every transaction a Boolean label, i.e. $\mathcal{D} = \{(t, T_t, a_t) \mid t \in \mathcal{T}, T_t \subseteq \mathcal{I}, a_t \in \{0, 1\}\}$.

Example 1. As an example consider the database in Fig. 1a. Ignoring the class label, this database can be represented as a set $\mathcal{D} = \{(1, \{1, 2, 3, 5\}), (2, \{1, 2, 4, 5\}), (3, \{1, 3, 4, 5\}), (4, \{1, 3, 4\})\}$.

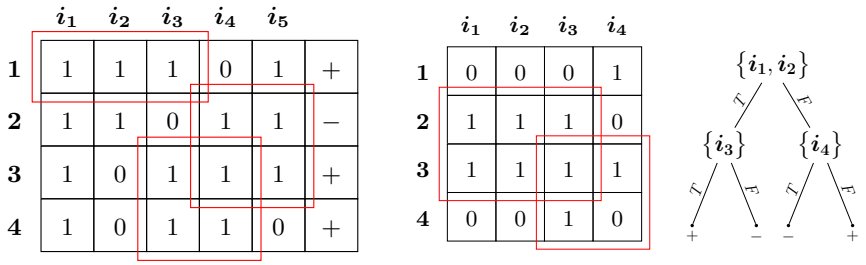
Several pattern set mining problems have been proposed on such Boolean data. We will first consider supervised settings. In **Concept Learning** [1] the aim is to discover a set of itemsets that characterizes the positive examples in a supervised dataset as well as possible.

Definition 1 (Concept Learning). ***Given*** *a Boolean supervised dataset $\mathcal{D}$,* ***find*** *a set of itemsets $C \subseteq 2^{\mathcal{I}}$, also referred to as concepts,* ***such that*** *$|C| = k$ and $error(\cup_{I \in C} cover(I))$ is minimal.*

Here, we define the cover and the error as follows:

- $cover(I) = \{t \mid (t, T_t) \in \mathcal{D} \wedge I \subseteq T_t\}$, the set of transactions that contain a given itemset;
- $error(T) = |\{t \in \mathcal{T} \mid (a_t = 1 \wedge t \notin T) \vee (a_t = 0 \wedge t \in T)\}|$, the number of examples not characterized correctly.

Example 2. Consider the concepts $C = \big\{\{1, 2, 3\}, \{3, 4\}, \{4, 5\}\big\}$; these are highlighted in Fig. 1a. Observation 2 is misclassified, so the error is 1.



(a) $\mathcal{D}_1$ (b) $\mathcal{D}_2$ (c) Tree

Fig. 1: Itemset databases showing **(a)** itemsets and its covers and **(b)** tiles. **(c)** An example of itemset-based tree.

In **Rule Learning** [5,20], we treat the patterns as rules that can also predict a negative class label. Restricting ourselves to the Boolean context, we can define the problem of learning Rule Lists as follows.

Definition 2 (Rule List Learning). ***Given*** *a Boolean supervised dataset $\mathcal{D}$,* ***find*** *a list of k rules $\mathcal{R} = \left\langle I^{(r)} \rightarrow l^{(r)} \right\rangle_{r=1}^{k}$, where $I^{(r)}$ is an itemset and $l^{(r)} \in \{0,1\}$ is a class label,* ***such that*** *$error(cover(\mathcal{R}))$ is minimal, and $I^{(k)} = \emptyset$, i.e., the kth rule serves as a default rule.*

Here, we define the cover of a rule list as the set of transactions for which the rule list predicts the positive class label, $cover(\mathcal{R}) = \{t \in \mathcal{T} \mid \exists (I^{(r)} \rightarrow 1) \in \mathcal{R} : I^{(r)} \subseteq T_t \wedge \neg(\exists r' < r : (I^{(r')} \rightarrow l^{(r')}) \in \mathcal{R} \wedge I^{(r')} \subseteq T_t)\}$.

Example 3. For the database of our running example (eg. 1), the rule list $\langle(\{1,2,3\} \rightarrow 1), (\{4,5\} \rightarrow 0), (\{3,4\} \rightarrow 1), (\emptyset \rightarrow 1)\rangle$ would have an error of 1: example 3 would be misclassified, as the first rule that applies to this example is the rule $\{4,5\} \rightarrow 0)$.

Itemset-based Decision Trees [10] are a generalization of rule lists. Essentially, they are decision trees in which each node tests for the presence of an itemset. A transaction (observation) will be put in the left-hand branch of an internal node v if the itemset $I^{(v)}$ is present, and in the right-hand branch if the itemset is absent. A *complete* decision tree is a tree that is filled completely till the lowest level.

Definition 3 (Learning Pattern-based Decision Trees). ***Given*** *a Boolean supervised dataset $\mathcal{D}$,* ***find*** *a set of k itemsets $\{I^{(r)}\}$, corresponding to internal nodes of a complete decision tree, and labels in $\{0,1\}$ for each leaf of the tree,* ***such that*** *the error of the predictions at the leaves of the tree is minimal.*

Example 4. Fig. 1c is an example of a pattern-based tree obtained from $\mathcal{D}_1$ with 2 as error because examples 3 & 4 are misclassified.

We will consider a number of pattern set mining settings on unsupervised data next. **Conceptual Clustering** [9] aims to cluster examples while also finding descriptions for these clusters. One possible definition of this problem is the following.

Definition 4 (Conceptual Clustering). ***Given*** *a Boolean unsupervised dataset $\mathcal{D}$,* ***find*** *k itemsets C,* ***such that*** *$\left|\left\{t \in \mathcal{T} : |\{I \in C : I \subseteq T_t\}| \neq 1\right\}\right|$ is minimal; hence, the number of transactions of the given dataset not covered by* `exactly` *one itemset is minimal.*

Example 5. In our running example (eg. 1) each itemset now describes a cluster (the target attribute of $\mathcal{D}_1$ is not taken into account). These clusters cover respectively transactions $\{1\}$, $\{3, 4\}$ and $\{2, 3\}$; the $\{3\}$ is an overlapping transaction.

Boolean matrix factorization [17] aims to describe the 1s in a database using two Boolean matrices, which can be seen as matrices describing itemsets and their occurrences.

Definition 5 (Boolean Matrix Factorization). ***Given*** *a Boolean database $\mathcal{D}$,* ***find*** *a Boolean matrix A of size $n \times k$ and a Boolean matrix B of size $k \times m$,* ***such that*** *$error(A \circ B, \mathcal{D})$ is minimal.*

Here $\circ$ is the Boolean matrix product, in which the matrix product is redefined such that $1 + 1 = 1$, and *error* is a function that calculates the number of cells in the two given matrices that mismatch.

Example 6. Fig.1b shows an example of two rectangles of which the rows and columns are identified using Boolean matrices A and B. The error is 2: cells $(1, 4)$ and $(4, 4)$ are not described correctly in this matrix decomposition.

In the next section, we show how these problems can be reformulated as parameter learning problems over *Boolean circuits*.

4 Reformulating Pattern Set Mining as Parameter Learning in Logical Circuits

In this section we will define the problem of *parameter learning in Boolean circuits*; subsequently, we will show that the learning problems identified in the previous section can all be cast as such parameter learning problems. We first define *Boolean circuits*.

Definition 6. *A* Boolean circuit *C is a* directed acyclic graph *$G(V, E)$ in which each node $v \in V$ of in-degree zero represents an input variable, only one node has an out-degree of zero, and each internal node represents a logical gate and is labeled with an operator from the set $\{\wedge, \vee, \neg\}$.*

Logical circuits can be seen as the graphical representation of a Boolean formula. Each internal node v with label $\wedge$ corresponds to an expression $v = v_1 \wedge \cdots \wedge v_n$, where $v_1, \ldots, v_n$ are v's children. For any assignment to the input variables, the Boolean circuit calculates a Boolean value for the output variable. Hence, we can see the Boolean circuit as a function from the Boolean input variables to $\{0, 1\}$.

We can define the parameter learning problem for a given Boolean circuit as follows.

Definition 7. *Given (1) a Boolean circuit C, (2) a partition of the input variables into two sets X and W, where W represents the* parameters *of the circuit, and (3) a Boolean supervised dataset $\mathcal{D}$ over $|X|$ items, the* parameter learning problem *is the problem of finding an assignment to the variables W such that*

$$\sum_{(t,T_t,a_t)\in\mathcal{D}} |C(W,T_t) - a_t| \tag{1}$$

is minimized, i.e., C fits a_t well. Here we assume that in passing T_t as a parameter to C, we set all variables in X to `True` *that are included in T_t.*

Below, we will show that the pattern set mining problems presented earlier can be represented as parameter learning problems for Boolean circuits, for well chosen architectures for C and, in some cases, representations of the input data.

However, before doing so, we will introduce some additional notation. We found that modeling the pattern set mining problems at the level of basic Boolean circuits is cumbersome. To simplify our modeling task, we will use an approach that is common in Boolean circuit design: we will add additional gates to our notation that can be seen as a shorthand notation for underlying, larger circuits.

Our first additional gate is $\circledvee$, which operates on two lists of inputs $\boldsymbol{w} = w_1, \ldots, w_n$ (each w_i corresponding to a variable in W) and $\boldsymbol{v} = v_1, \ldots, v_n$ (each v_i corresponding to a variable in V), and which can be understood as a shorthand notation for

$$\circledvee(\boldsymbol{w}, \boldsymbol{v}) \equiv (w_1 \wedge v_1) \vee (w_2 \wedge v_2) \vee \cdots \vee (w_n \wedge v_n).$$

The idea is that the parameters $w_1, \ldots, w_n$ indicate which of the inputs of the $\vee$ should be taken into account.

Similarly, we define $\circledwedge$ as a shorthand notation for

$$\circledwedge(\boldsymbol{w}, \boldsymbol{v}) \equiv (\neg w_1 \vee v_1) \wedge (\neg w_2 \vee v_2) \wedge \cdots \wedge (\neg w_n \vee v_n),$$

where the parameters $w_1, \ldots, w_n$ indicate which of the inputs of the $\wedge$ should be taken into account.

In this paper, we will use a graphical notation for Boolean circuits. We will illustrate this notation first on the problem of learning rule lists.

Fig. 2 (left) shows the Boolean circuit for learning rule lists using the shorthand notation, for a dataset with 3 items and a rule list of at most two rules, plus the default rule. In the shorthand notation, we use an $\wedge$ symbol (respectively, $\vee$ symbol) with dotted incoming edges to represent the $\circledwedge$ (respectively, $\circledvee$) gate. The dotted edges indicate that we have a parameter for each such edge, indicating whether or not to take into account that edge. Hence, in this diagram the input variables of the circuit representing parameters are not explicitly included. We can distinguish three types of layers in this circuit:

Layer 1 represents the problem of selecting which items are included in each rule; the output nodes of this layer can be seen as indicators for the absence or presence of a rule in a data instance;

Layer 3 represents the problem of selecting the class label for the two rules, as well as the class label for the default rule;

Layer 2 expresses the order dependencies between the rules; it ensures that the second rule in the rule list will only be used for prediction if the first rule does not match, and the default rule will only be used if the previous two rules did not match.

Fig. 2 (right) show the full circuit representation with explicit binary decision variables encoding the rules to be discovered that can be retrieved as follows: rule 1 is $\{i \mid i \in \{1,\ldots,3\} \wedge I_i^{(1)} = 1\} \rightarrow L^{(1)}$, rule 2 is $\{i \mid i \in \{1,\ldots,3\} \wedge I_i^{(2)} = 1\} \rightarrow L^{(2)}$ and the last (default rule) is $\emptyset \rightarrow L^{(3)}$. The inputs of the circuit are $X = \{i_1, i_2, i_3\}$. The variables $I_1^{(1)} \ldots I_3^{(2)}, L^{(1)} \ldots L^{(3)}$ are shared among all the transactions to impose that all transactions are classified with a unique rule list.

Similarly, for the other problems introduced in section 3, we can introduce Boolean circuits to formalize these mining and learning problems. The architecture of these circuits is illustrated in Figure 3, for small examples.

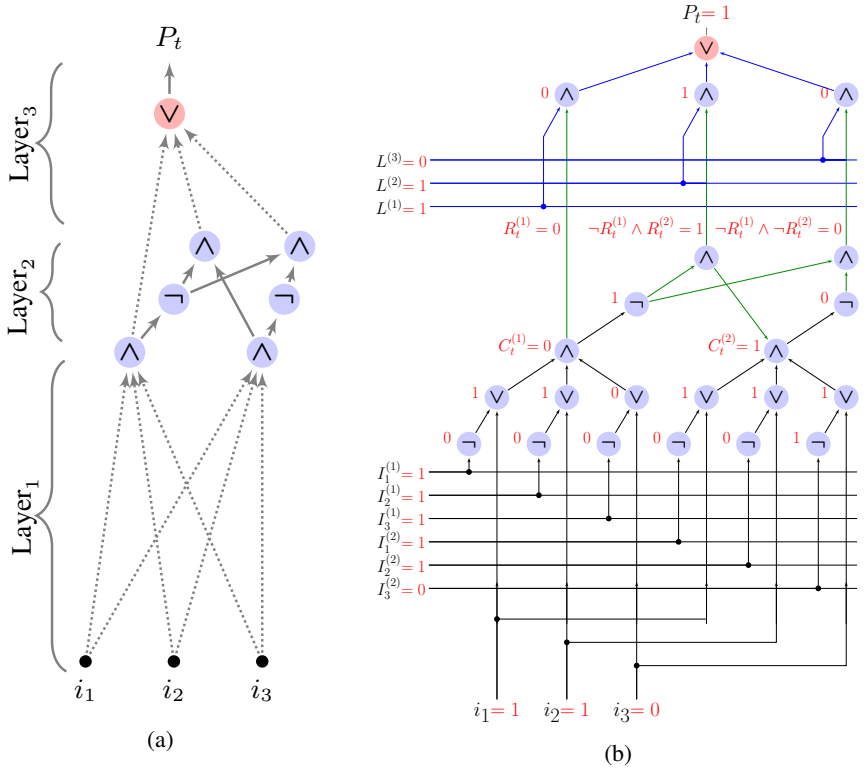
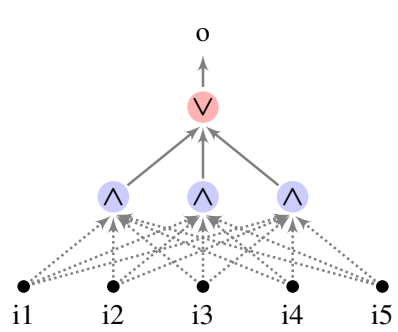
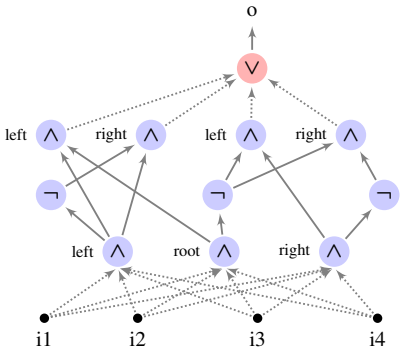


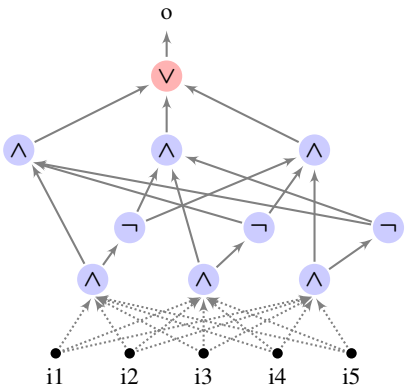
Fig. 2: Architecture of Boolean circuit for rule learning with $m = 3$ items and $k = 3$ rule list size: **a)** general representation **b)** full representation with decision variables (dotted arrows represent optional decisions and solid arrows mandatory decisions).



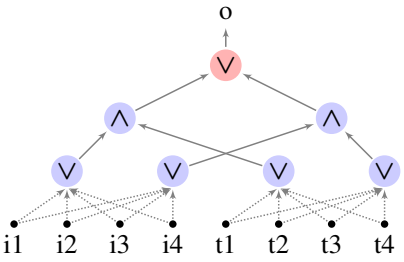
(a) Concept Learning



(b) Learning Itemset-based Decision Trees



(c) Conceptual Clustering



(d) Boolean Matrix Factorisation

Fig. 3: Architectures of Boolean circuits for the other studied problems (dotted arrows represent optional decisions and solid arrows mandatory decisions).

For **concept learning** and **itemset-based decision trees** the data used for training is identical to that used for rule list learning. The difference between concept learning and rule list learning is that in concept learning the head of the rules is fixed and there is no order between the rules. The decision tree learning problem illustrated is for decision trees composed of 3 itemsets: one for the root and one for each of the children of the root. In the Boolean circuit every child of the root corresponds to one of the leaves of the decision tree. The extension towards perfect trees of larger depth is relatively straightforward.

Conceptual clustering in an unsupervised problem setting; however, in our parameter learning setting we need to provide a label for every training instance. We address this by giving every instance in the training data the label 1. The idea in this circuit is that we predict 1 for an example iff there is exactly one itemset that matches it; if no itemset, or two or more itemsets match it, we predict 0. As a result, the error score for this circuit corresponds to counting the number of examples not exactly in one cluster.

Boolean matrix factorization is an unsupervised setting as well. The input data that we give to the Boolean circuit is different here. Given the original data $\mathcal{D}_{ti}$, we create a new dataset as follows:

$$\Big\{\big((t,i),\{t,i\},\mathcal{D}_{ti}\big) \mid t \in \mathcal{T}, i \in \mathcal{I}\Big\},$$

that is, we create a dataset in which each entry of the original matrix is an example; every new example consists of two items: one representing the original item, the other the original transaction identifier. In the lowest layer of the Boolean circuit, both transaction sets and itemsets are identified; the layer on top represents the Boolean matrix product. It can be shown that this model is equivalent to the original learning problem a well.

5 Generic Solving Framework

The question arises now how to solve these parameter learning problems. The benefit of our approach is that it allows for the use of alternative solvers. In this paper, we will consider two such approaches: one is the use of a greedy algorithm; the other is the use of Mixed Integer Programming (MIP) solvers.

5.1 Solving using Greedy algorithm

Greedy algorithms are among the most scalable algorithms for the pattern set mining algorithms studied in this work. Indeed, for rule learning tasks these are the most common type of algorithm, as in practice, the solution found by such algorithms is already of decent quality. The parameters of the Boolean circuit that minimize the error (1) can also be found greedily.

Algorithm 1 shows a greedy algorithm. This algorithm receives the Boolean circuit (together with its partition of inputs into two sets W and X) and the database $\mathcal{D}$. We will represent the values of the parameters by listing the subset of parameters $\mathcal{P} = \{1, 2, \ldots, |W|\}$ that take the value 1. By abuse of notation, we hence treat a vector of assignments to the variables in W as a subset of parameters $\mathcal{P}$. The algorithm starts with an empty set of parameters W and then iteratively identifies the parameter for which a flip from the value 0 to the value 1 minimizes the error (Line 5). Once a local optimal parameter is found, W is updated accordingly in line 6. The process repeated until either *(i)* a solution better than W cannot be found (line 7) or *(ii)* all variables have been fixed to the value 1, or *(iii)* the minimum error is 0.

The set $\mathcal{P}$ represents all the dotted edges in our graphical representation of the learning problems. An empty list W indicates that no edge is selected. Taking a decision (include or not) corresponds to flipping the parameter value.

While we could apply this greedy algorithm on all parameters of the circuit, we perform an optimization when the root node of the circuit consists of a $\vee$ node, such as in **rule learning** and **itemset-based decision trees**, and for every transaction only one of the children takes the value `True`. For such circuits it can be shown that the optimal choice for the children of the root can easily be calculated from the choices below those children. Hence, we do not perform a greedy search over this set of children.

Algorithm 1: $Greedy(C, \mathcal{D})$

Method error (W**: Assignment to the parameters,** C**: Logical Circuit,** $\mathcal{D}$**: list of Transactions)**
  return $\sum_{(t,T_t,a_t)\in\mathcal{D}} |C(W,T_t) - a_t|$
$W \leftarrow \emptyset$ `minErr` $\leftarrow$ error$(W, \mathcal{D})$
do
  $w^* \leftarrow \underset{i\in\mathcal{P}\setminus W}{\operatorname{argmin}}\ \text{error}(W \cup \{i\}, C, \mathcal{D})$
  if error$(W \cup \{w^*\}, C, \mathcal{D}) <$ `minErr` **then** $W \leftarrow W \cup \{w^*\}$ ▷ *Add* w^* *to* W ;
  else break;
while $\mathcal{P} \setminus W \neq \emptyset \wedge$ `minErr` > 0; ▷ *Stop if no more decision or* `minErr` $= 0$
return W

For some problems, such as **conceptual clustering** and **Boolean matrix factorization**, we found that it can be beneficial to start from an assignment that puts all variables in W at the value 1. This can be emulated by putting a $\neg$ node between every parameter and the nodes that it is connected to.

5.2 Solving using MIP

The key idea in this approach is to map the parameter learning problem to an optimization problem defined on integer variables and linear constraints. An additional integer variable is introduced for each node in the circuit and each gate is modeled using a set of linear constraints as follows:

– if a node y is an $\wedge$-*gate* of x_i nodes:

$$y = x_1 \wedge x_2 \wedge \cdots \wedge x_m \equiv \begin{cases} y \leq x_i \quad \forall i \in [1, m] \\ y \geq \Sigma_i x_i - (m-1) \\ y \geq 0 \end{cases}$$

– if a node y is an $\vee$-*gate* of x_i nodes:

$$y = x_1 \vee x_2 \vee \cdots \vee x_m \equiv \begin{cases} y \geq x_i \quad \forall i \in [1, m] \\ y \leq \Sigma_i x_i \\ y \leq 1 \end{cases}$$

– finally, if a node y is a $\neg$-*gate* of x node:

$$y = \neg x \equiv y = 1 - x$$

We make a copy of the circuit for every example in the training data, fixing the corresponding inputs of the circuit to the values in the training example; the parameters are variables that the MIP solver will search over. For every training example, we will include the output v of the circuit in the optimization criterion, using v if the expected output is 0 and $(1 - v)$ if the expected output is 1. We minimize this error.

Algorithm 2: DSL to solve a rule learning problem with the architecture of Fig.2a and $\mathcal{D}$ in MIP and Greedy algorithm

```
1  ▷ Building the network
2  L_1 ← InputLayer(m = 3)
3  L_2 ← AndSelectionLayer(L_1, k = 3)
4  L_3 ← NotLayer(L_2)
5  L_4 ← Layer(L_2[1], And(L_3[1], L_2[2]), And(L_3[1], L_3[2]))
6  L_5 ← OrSelectionLayer(L_4)
7  N ← L_5.network()
8  ▷ Load inputs and parameters
9  X ← getDB(D)   y ← getAttr(D)   ŷ ← L_5[0]
10 obj ← 1 − y.ŷ − (1 − y)(1 − ŷ)                    ▷ Objective function
11 ▷ Defining the procedures
12 greedy ← X into N using Greedy.solver minimizing obj
13 stats ← greedy.run()
14 mip ← X into N using MIP.solver minimizing obj
15 stats ← mip.run()
```

Note that this gives a generic approach for solving the mining and learning problems discussed earlier using MIP solvers. There is already a literature on modeling some of these individual problems in MIP (see [4,22,23,24,27]; our approach provides a more general approach to modeling such problems.

6 Unified Modeling and Solving Language

In our vision, creating parameter learning problems for Boolean circuits can be seen as a programming task. The main benefit of our framework is that its modeling language is very similar to that of Deep Learning toolkits, and hence familiar to machine learning researchers.

Algorithm 2 is an example of an implementation of the rule list learning problem based on the architecture of Fig. 2a. From line 2 to 7 the Boolean circuit is defined by using *macro-functions* such as *AndSelectionLayer* which represents all the operations of Layer 1 in Fig. 2b. At line 10, we define the error function and at lines 12 and 14 the different solvers, which are launched in lines 13 and 15.

7 Learning Classifiers based on Soft Rules

While we focused our modeling framework on Boolean parameters only, of interest can also be combinations of Boolean parameters with other types of parameters. We will study one first possible such combination here and will leave other combinations as future work.

In traditional learning, the interpretation of the conditions in a rule is typically *conjunctive*: all conditions in a rule need to be met. However, this conjunctive interpretation

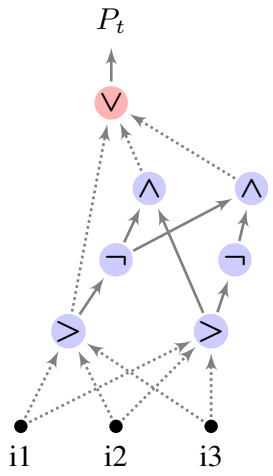


Fig. 4: Learning soft rule lists

of rules can also be considered a limitation. More freedom would be allowed in rule-based classifiers in which we require to match a certain number of conditions, but not necessarily all.

We can model such problems by adding a *parameterized soft gate* to our network. The new *parameterized soft gate* requires that at least α inputs should be true, where α is also a parameter that needs to be learned. More formally, this gate operates on three inputs, and has the following semantics: $\circledS(\boldsymbol{v}, \boldsymbol{w}, \alpha) \equiv \sum_i v_i w_i \geq \alpha$, where $v_1, \ldots, v_n$ indicate Boolean input nodes, and $w_1, \ldots, w_n$ are Boolean indicators representing whether or not to take into account the ith input. Note that $\circledS(\boldsymbol{v}, \boldsymbol{w}, 1) = \circledV(\boldsymbol{v}, \boldsymbol{w})$: the gate generalizes the $\circledV$-gate. Furthermore, $\circledS(\boldsymbol{v}, \boldsymbol{w}, n) \equiv v_1 \wedge \cdots \wedge v_n$.

We can use this gate to define a rule learning problem in which each rule does specify not only conditions, but also the minimum number of conditions that need to be satisfied in order for the rule to apply. This is illustrated in Figure 4. This model has a higher level of expressivity than traditional rule learning models: by fixing the parameters α to a sufficiently high level, we can still enforce that all conditions need to be satisfied for a rule to apply.

We can learn the parameters of such gates both using MIP and using greedy algorithms. In MIP, we exploit the fact that $v_i w_i \equiv v_i \wedge w_i$. Let $u_i \equiv v_i \wedge w_i$; then we implement the gate by first calculating the u_is using the representation for $\wedge$ discussed earlier and then using these u_is as follows:

$$y = \circledS(\boldsymbol{v}, \boldsymbol{w}, \alpha) = \begin{cases} \sum_{i=1}^{n} u_i - \alpha + (1-y)n & \geq 0 \\ \sum_{i=1}^{n} u_i - \alpha - yn + 1 & \leq 0. \end{cases}$$

Note that also here we copy the circuit for each example in the data; the u_is are calculated for each example separately. However, the parameters α and w_i are shared among all examples.

8 Experiments

In this section, we evaluate our framework from three perspectives: (i) the predictive power of the classifiers learned compared to other classifiers, (ii) the sensitivity of the pattern sets identified w.r.t. the variation of parameters (like k),(iii) the efficiency of the framework (using CPU time). All experiments were run in the JVM with maximum memory set to 8GB on PCs with Intel Core i5 64bit processor (2.7GHz) and 16GB of RAM running MAC OS 10.13.3. Execution time is limited to one hour.

Datasets and existing classifiers. We use data from the CP4IM[1] repository. Statistics of these datasets are reported in Table 1a. We compare with the following methods: (i) Popular tree-based and neural network-based classifiers such as Random Forests (RF), decision trees (C4.5) and neural networks (NN) from the *scikit-learn* library (using default settings); (ii) a rule-based learner: Probabilistic Rule List (PRL) [2] and a k-pattern set miner [14,12] (KPATT) for the concept learning task, in which the concept learning problem is solved as a global optimization problem in a CP solver.

We denote our approaches by $X4Yz$, where $X \in \{$*MIP, G*$\}$ is the solving strategy and can be either *MIP* or *G*(greedy), $Y \in \{$*CL, RL, PDT, BMF, CC* $\}$ represents the pattern set mining problem solved and can be *CL* (concept learning), *RL* (rule learning), *PDT* (Pattern-based Decision Trees), *BMF* (Boolean Matrix factorization) or *CC* (conceptual clustering). We use $z \in \{+\alpha, -\alpha\}$ to indicate whether or not the soft gates are used (see sect. 7).

Comparison of Model Quality. Table 1e) shows the average accuracy evaluated using *stratified 10-fold cross-validation* on test data. Note that in some cases, within the time allocated the MIP solver could not prove optimality. We use the best pattern set found within the allocated amount of time. In these experiments, the number of patterns is fixed arbitrarily to $k = 5$.

RF and Neural Networks perform better than the rule-based methods, but our introduction of soft rules (sect. 7) seems to improve the accuracy of rule-based methods in most cases over both the training and the test tests. The performance on test data of the greedy algorithm is sometimes worse and sometimes better than that of the MIP-based algorithm, although on concept learning the performance of the greedy algorithm is not satisfactory. For many cases where optimality could not be proven within the allocated time, the optimality gap is less than 10%.

Running time comparison. The greedy approach is generally very efficient and outputs results in a few seconds, so we only report the execution time of the MIP approach in Table 1. As one can see, in many cases, the timeout is reached with a gap from the optimum smaller than 10%, with a few exceptions (41%, 29%). Our approaches outperform KPATT on small databases and on (relatively) large databases KPATT fails to find a solution within the allocated amount of time.

MIP approaches are highly dependent on the number of variables and the number of constraints, which are $\mathcal{O}(k \times |\mathcal{T}| \times |\mathcal{I}|)$ and $\mathcal{O}\big(k \times |\mathcal{T}| \times (|\mathcal{I}| + k)\big)$ respectively. However, the MIP pre-solving is able to drastically reduce the number of variables and

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

Table 1: Experiments for pattern set mining problems over several datasets with $k = 5$ ("-" means the process stopped before a solution was found, either due to a *out of memory/timeout exception* in the pre-solving step of the MIP solver, or before the CP solver used in KPATT found a solution).

methods	Audi.	Aust.	HeCl.	Hepa.	KrKp.	Lymp.	Mush.	PrTu.	Soyb.	Spli.	TTT.	Vote	Zoo
	a) Dataset Features												
$\|\mathcal{T}\|$	216	653	296	137	3196	148	8124	336	630	3190	958	435	101
$\frac{\|\{t\in\mathcal{T}\|a_t=1\}\|}{\mathcal{T}}$	0.26	0.55	0.54	0.81	0.52	0.55	0.52	0.24	0.15	0.52	0.65	0.61	0.41
$\|\mathcal{I}\|$	148	125	95	68	74	68	119	31	50	287	27	48	36
	b) Accuracies over training sets												
MIP4CL-α	**1.0**	**0.92**	0.89	0.98	0.87	0.97	0.55	0.89	0.97	-	**0.90**	0.98	**1.0**
MIP4CL+α	**1.0**	0.91	**1.0**	**1.0**	0.93	**1.0**	**1.0**	**0.91**	**1.0**	**0.85**	0.77	**1.0**	**1.0**
G4CL	0.73	0.45	0.46	0.19	0.47	0.46	0.49	0.76	0.86	0.48	0.35	0.37	0.59
KPATT	**1.0**	-	-	-	-	-	-	0.89	0.97	-	0.82	-	**1.0**
MIP4RL-α	**1.0**	-	-	**1.0**	-	0.95	-	**0.89**	0.96	-	0.81	0.98	**1.0**
MIP4RL+α	**1.0**	-	0.97	**1.0**	-	**1.0**	**1.0**	0.87	**0.98**	-	**0.83**	**1.0**	**1.0**
G4RL	0.98	**0.86**	0.83	0.89	**0.94**	0.86	0.98	0.84	0.86	**0.84**	0.70	0.96	**1.0**
MIP4PDT-α	**1.0**	-	0.89	**1.0**	-	0.96	-	0.89	0.94	-	-	0.97	**1.0**
MIP4PDT+α	**1.0**	-	**1.0**	**1.0**	-	**1.0**	**1.0**	-	**0.99**	-	-	**1.0**	**1.0**
G4PDT	0.99	**0.86**	0.82	0.89	**0.94**	0.88	**1.0**	0.84	0.86	**0.84**	**0.76**	0.96	**1.0**
	c) Gap (%) for MIP over training sets												
MIP4CL-α	*	0.08	0.12	0.02	0.12	0.03	0.41	0.07	0.03	-	0.11	0.02	*
MIP4CL+α	*	0.10	*	*	0.07	*	*	0.05	*	0.16	0.29	*	*
MIP4RL-α	*	-	-	*	-	0.05	-	0.08	0.05	-	0.24	0.02	*
MIP4RL+α	*	-	0.03	*	-	*	*	0.10	0.02	-	0.20	*	*
MIP4PDT-α	*	-	0.12	*	-	0.04	-	0.08	0.06	-	-	0.03	*
MIP4PDT+α	*	-	*	*	-	*	*	-	0.01	-	-	*	*
	d) Running time (in second) - TO$\equiv$ Timeout												
MIP4CL-α	26.09	TO	TO	TO	TO	TO	TO	TO	TO	-	TO	TO	1.65
MIP4CL+α	**5.81**	TO	**2682.90**	**1.99**	TO	**2.50**	**251**	TO	**915.09**	TO	TO	**17.32**	**0.66**
KPATT	20	-	-	-	-	-	-	TO	1730.30	-	TO	-	3.29
MIP4RL-α	45.45	-	-	31.65	-	TO	-	TO	TO	-	TO	TO	1.70
MIP4RL+α	**7.73**	-	TO	**9.00**	-	**5.72**	**1103.95**	TO	TO	-	TO	**146.90**	**0.69**
MIP4PDT-α	51.61	-	TO	3038.61	-	TO	-	TO	TO	-	-	TO	2.26
MIP4PDT+α	**12.25**	-	**757.20**	**9.27**	-	**9.90**	TO	-	TO	-	-	**1265.56**	**1.05**
MIP4CC-α	42.04	244.73	46.53	7.58	1072.34	**1.91**	-	16.77	123.81	-	153.56	**5.71**	2.57
MIP4CC+α	**11.63**	**3.34**	**9.86**	2.27	**504.26**	**2.32**	**1118.64**	**7.26**	**1.62**	**443.68**	**22.56**	7.38	**0.60**
MIP4BMF-α	-	-	-	556.20	-	697.97	-	1454.71	-	-	-	-	127.15
MIP4BMF+α	-	-	-	**542.93**	-	**65.20**	-	**1446.05**	-	-	-	-	**123.20**
	e) Accuracies over test sets												
MIP4CL-α	0.87	**0.85**	**0.77**	**0.79**	0.86	0.69	0.55	**0.83**	**0.98**	-	**0.94**	**0.98**	**1.0**
MIP4CL+α	**0.91**	0.76	0.70	**0.79**	**0.89**	**0.75**	**1.0**	0.8	0.89	**0.82**	0.75	0.93	**1.0**
G4CL	0.77	0.48	0.42	0.21	0.51	0.38	0.45	0.71	0.84	0.47	0.34	0.49	0.64
MIP4RL-α	0.87	-	-	0.79	-	0.62	-	**0.89**	**0.95**	-	**0.86**	**1.0**	**1.0**
MIP4RL+α	0.87	-	**0.8**	**0.93**	-	0.62	**1.0**	0.8	0.89	-	0.81	0.98	**1.0**
G4RL	**1.0**	**0.94**	0.65	0.79	**0.95**	**0.81**	0.99	0.76	0.84	**0.85**	0.72	0.96	**1.0**
MIP4PDT-α	0.83	-	0.8	**0.86**	-	0.75	-	**0.77**	0.92	-	-	0.95	**1.0**
MIP4PDT+α	0.91	-	**0.83**	**0.86**	-	0.69	**1.0**	-	**0.95**	-	-	0.93	**1.0**
G4PDT	**1.0**	**0.94**	0.65	0.79	**0.95**	**0.81**	**1.0**	0.76	0.84	**0.85**	**0.77**	**0.96**	**1.0**
PRL	0.87	-	-	0.71	-	0.62	-	0.26	0.83	-	-	0.64	**1.0**
C4.5	0.94	0.81	0.75	0.72	**0.98**	0.84	0.97	0.75	0.91	0.93	0.82	0.95	0.97
RF	0.95	0.83	0.76	**0.82**	0.96	0.85	0.97	**0.79**	**0.95**	0.94	**0.87**	0.96	**1.0**
NN	**1.0**	**0.91**	**0.87**	0.79	0.96	**1.0**	**0.99**	0.71	0.84	**0.95**	0.79	**0.98**	**1.0**

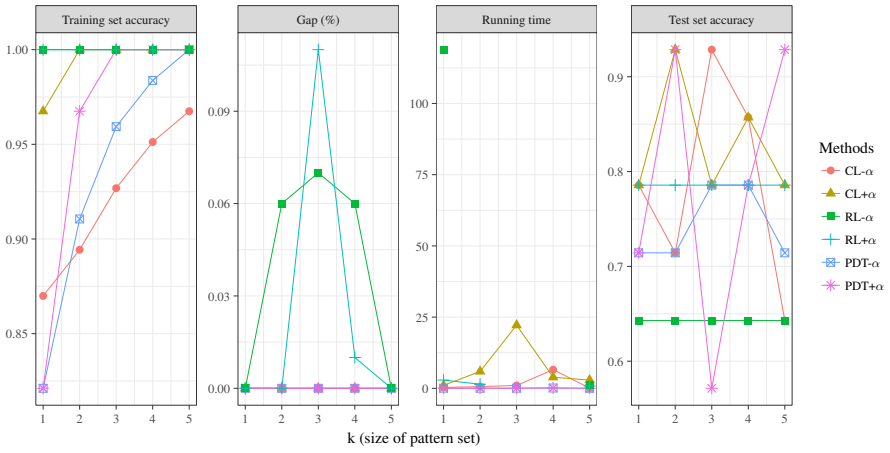


Fig. 5: Sensitivity to the parameter k of our approches over Hepa dataset (time limit = 600s) .

constraints in some cases. For example, the initial numbers of variables $(424,945)$ and constraints $(710,632)$ in the Audi dataset were reduced by more than 97%.

Sensitivity to the parameter k. Figure 5 shows the accuracy on training and test sets, the gap, and the execution time by varying k for the Hepa database; results on other datasets are similar. As can be seen, increasing k improves the accuracy on the training set. On the test set the outcome depends strongly either on whether optimality has been proven or not (as evidenced by the gap-plot) or overfitting on the training set.

9 Conclusion

Motivated and inspired by the high level modeling language offered by deep learning frameworks, this work proposed a unified modeling framework for various k-pattern set mining problems (concept learning, rule list learning, pattern-based decision trees, conceptual clustering and Boolean matrix factorization). The modeling language is independent from the optimization technology. We have shown that possible extensions of the language are possible and yield promising results, such as the soft gates. We have shown experimentally that despite the genericity of our framework, the performance of the approach is competitive to that of existing traditional learning approaches, and outperforms an earlier CP-based approach for pattern set mining.

Many future studies are possible. First, alternative optimization approaches are of interest, such as based on meta-heuristics (such as large neighborhood search) and gradient-based approaches, in particular those for learning *binarized neural networks* [15]. Furthermore, other links to deep learning can be explored further, for instance, in mixed networks that combine discrete and continuous components. In this work we did not restrict the form of Boolean circuit used; by adding restrictions on the form of the Boolean circuit, such as that the Boolean circuit is in decomposable negation normal form [7], it may be possible to build more optimized algorithms.

References

1. Angluin, D.: Queries and concept learning. Machine Learning **2**(4), 319–342 (1987)
2. Aoga, J.O.R., Guns, T., Nijssen, S., Schaus, P.: Finding probabilistic rule lists using the minimum description length principle. In: Soldatova, L.N., Vanschoren, J., Papadopoulos, G.A., Ceci, M. (eds.) Discovery Science - 21st International Conference, DS 2018, Limassol, Cyprus, October 29-31, 2018, Proceedings. Lecture Notes in Computer Science, vol. 11198, pp. 66–82. Springer (2018)
3. Aoga, J.O.R., Guns, T., Schaus, P.: Mining time-constrained sequential patterns with constraint programming. Constraints **22**(4), 548–570 (2017)
4. Bertsimas, D., Dunn, J.: Optimal classification trees. Machine Learning **106**(7), 1039–1082 (2017)
5. Clark, P., Niblett, T.: The CN2 induction algorithm. Machine Learning **3**, 261–283 (1989)
6. Coquery, E., Jabbour, S., Saïs, L., Salhi, Y.: A SAT-based approach for discovering frequent, closed and maximal patterns in a sequence. In: ECAI (2012)
7. Darwiche, A.: Compiling knowledge into decomposable negation normal form. In: Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence. pp. 284–289. IJCAI '99, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1999)
8. De Raedt, L., Zimmermann, A.: Constraint-based pattern set mining. In: Proceedings of the Seventh SIAM International Conference on Data Mining, April 26-28, 2007, Minneapolis, Minnesota, USA. pp. 237–248 (2007)
9. Fisher, D.H., Langley, P.: Approaches to conceptual clustering. In: Joshi, A.K. (ed.) Proceedings of the 9th International Joint Conference on Artificial Intelligence. Los Angeles, CA, USA, August 1985. pp. 691–697. Morgan Kaufmann (1985)
10. Gay, D., Selmaoui, N., Boulicaut, J.: Pattern-based decision tree construction. In: Second IEEE International Conference on Digital Information Management (ICDIM), December 11-13, 2007, Lyon, France, Proceedings. pp. 291–296. IEEE (2007)
11. Guns, T., Dries, A., Tack, G., Nijssen, S., De Raedt, L.: Miningzinc: A modeling language for constraint-based mining. In: Twenty-Third International Joint Conference on Artificial Intelligence (2013)
12. Guns, T., Nijssen, S., De Raedt, L.: Evaluating pattern set mining strategies in a constraint programming framework. In: Huang, J.Z., Cao, L., Srivastava, J. (eds.) Advances in Knowledge Discovery and Data Mining - 15th Pacific-Asia Conference, PAKDD 2011, Shenzhen, China, May 24-27, 2011, Proceedings, Part II. Lecture Notes in Computer Science, vol. 6635, pp. 382–394. Springer (2011)
13. Guns, T., Nijssen, S., De Raedt, L.: Itemset mining: A constraint programming perspective. Artificial Intelligence **175**(12-13), 1951–1983 (2011)
14. Guns, T., Nijssen, S., De Raedt, L.: k-Pattern set mining under constraints. IEEE Transactions on Knowl. and Data Eng. **25**(2), 402–418 (Feb 2013)
15. Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., Bengio, Y.: Binarized neural networks. In: Lee, D.D., Sugiyama, M., Luxburg, U.V., Guyon, I., Garnett, R. (eds.) Advances in Neural Information Processing Systems 29, pp. 4107–4115. Curran Associates, Inc. (2016), http://papers.nips.cc/paper/6573-binarized-neural-networks.pdf
16. Kemmar, A., Lebbah, Y., Loudni, S., Boizumault, P., Charnois, T.: Prefix-projection global constraint and top-k approach for sequential pattern mining. Constraints **22**(2), 265–306 (2017)
17. Lam, H.T., Pei, W., Prado, A., Jeudy, B., Fromont, É.: Mining top-k largest tiles in a data stream. In: Calders, T., Esposito, F., Hüllermeier, E., Meo, R. (eds.) Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2014, Nancy, France, September 15-19, 2014. Proceedings, Part II. Lecture Notes in Computer Science, vol. 8725, pp. 82–97. Springer (2014)

18. Lazaar, N., Lebbah, Y., Loudni, S., Maamar, M., Lemière, V., Bessiere, C., Boizumault, P.: A global constraint for closed frequent pattern mining. In: International Conference on Principles and Practice of Constraint Programmingn (CP). pp. 333–349. Springer (2016)
19. Meo, R., Psaila, G., Ceri, S.: A new sql-like operator for mining association rules. In: Vijayaraman, T.M., Buchmann, A.P., Mohan, C., Sarda, N.L. (eds.) VLDB'96, Proceedings of 22th International Conference on Very Large Data Bases, September 3-6, 1996, Mumbai (Bombay), India. pp. 122–133. Morgan Kaufmann (1996)
20. Michalski, R.S.: On the quasi-minimal solution of the general covering problem (1969)
21. Negrevergne, B., Guns, T.: Constraint-based sequence mining using constraint programming. In: CPAIOR 2015. pp. 288–305. Springer (2015)
22. Ouali, A., Loudni, S., Lebbah, Y., Boizumault, P., Zimmermann, A., Loukil, L.: Efficiently finding conceptual clustering models with integer linear programming. In: Kambhampati, S. (ed.) Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016. pp. 647–654. IJCAI/AAAI Press (2016)
23. Ouali, A., Zimmermann, A., Loudni, S., Lebbah, Y., Crémilleux, B., Boizumault, P., Loukil, L.: Integer linear programming for pattern set mining; with an application to tiling. In: Kim, J., Shim, K., Cao, L., Lee, J., Lin, X., Moon, Y. (eds.) Advances in Knowledge Discovery and Data Mining - 21st Pacific-Asia Conference, PAKDD 2017, Jeju, South Korea, May 23-26, 2017, Proceedings, Part II. Lecture Notes in Computer Science, vol. 10235, pp. 286–299 (2017)
24. Rudin, C., Ertekin, Ş.: Learning customized and optimized lists of rules with mathematical programming. Mathematical Programming Computation **10**(4), 659–702 (2018)
25. Schaus, P., Aoga, J.O.R., Guns, T.: Coversize: A global constraint for frequency-based itemset mining. In: Beck, J.C. (ed.) Principles and Practice of Constraint Programming, CP'17, Proceedings. Lecture Notes in Computer Science, vol. 10416, pp. 529–546. Springer (2017)
26. Sejnowski, T.J.: The deep learning revolution. MIT Press (2018)
27. Verwer, S., Zhang, Y.: Learning optimal classification trees using a binary linear program formulation. In: 33rd AAAI Conference on Artificial Intelligence (2019)