Intelligent Data Analysis 13 (2009) 109–133
DOI 10.3233/IDA-2009-0358
IOS Press

Mining constraint-based patterns using automatic relaxation

Arnaud Soulet[a,*] and Bruno Crémilleux[b]
[a]*LI, Université François Rabelais de Tours, 3 Place Jean Jaurès, F-41029 Blois, France*
[b]*GREYC-CNRS, Université de Caen, Campus Côte de Nacre, F-14032 Caen Cédex, France*

Abstract. Constraint-based mining is an active field of research which is a necessary step to achieve interactive and successful KDD processes. The limitations of the task lies in languages being limited to describe the mined patterns and the ability to express varied constraints. In practice, current approaches focus on a language and the most generic frameworks mine individually or simultaneously a monotone and an anti-monotone constraints. In this paper, we propose a generic framework dealing with any partially ordered language and a large set of constraints. We prove that this set of constraints called primitive-based constraints not only is a superclass of both kinds of monotone ones and their boolean combinations but also other classes such as convertible and succinct constraints. We show that the primitive-based constraints can be efficiently mined thanks to a relaxation method based on virtual patterns which summarize the specificities of the search space. Indeed, this approach automatically deduces pruning conditions having suitable monotone properties and thus these conditions can be pushed into usual constraint mining algorithms. We study the optimal relaxations. Finally, we provide an experimental illustration of the efficiency of our proposal by experimenting it on several contexts.

Keywords: Constraint-based mining, relaxation, monotonicity, virtual patterns

1. Introduction

Many data mining methods have been designed to support constraint-based pattern discovery. A constraint expresses by a declarative way the viewpoint of the analyst and guarantees the a priori interestingness of the extracted patterns. This task is important to achieve interactive and successful Knowledge Discovery in Databases (KDD) processes. Constraint-based patterns are used in a lot of domains as medicine [25], business data analysis [2], XML document analysis [42]. The form of the constraints allows users to describe the rules that they would like to uncover, thereby making the data mining process more effective. Constraints can also include interestingness measures as statistical [30] or others as the area (cf. Section 2.4) to capture relationships in data. This is very useful in real-world applications. For instance, in genomics, constraint-based knowledge discovery supported by the area measure has demonstrated that it allows to generate new biological hypotheses with clinical implications [23].

A lot of works address the design of sound and complete solvers on several classes of constraints. The soundness and completeness of the extraction ensure that the collection of extracted patterns is

*Corresponding author. E-mail: arnaud.soulet@univ-tours.fr.

respectively correct and exhaustive with respect to the constraint. It remains a challenge due to the huge size of the search space which has to be explored. In practice, works are devoted to a specific language describing the patterns (e.g., itemsets for transactional data [2]) and to a class of constraints (e.g., monotone [27], convertible [33], succinct [31]) or a combination of a monotone and an anti-monotone constraints [26]. In this paper, we cope with these limitations by designing a generic framework to define constraints and automatically deduce pruning conditions which can be efficiently pushed into usual constraint mining algorithms. The key ideas are to use a set of primitives which can be extended to get a broad spectrum of constraints and a relaxation approach based on virtual patterns to take into account both the specificities of the data and the original constraint q given by the user to automatically deduce outstandingly pruning conditions. The relaxation enables us to approximate the collection of patterns satisfying q by a larger collection corresponding to a solution space of a slightly weaker constraint. Then, we benefit suitable monotonicity properties from the relaxed constraint which can be exploited by usual constraint mining algorithms to output the correct and complete collection of constrained patterns. The approach strongly differs from the usual methods which seek to straightforwardly push q.

The contribution of this paper is twofold. First we propose a general framework (called primitive-based framework) for constraint-based pattern discovery. This framework deals with any partially ordered language describing the patterns (e.g., itemsets, sequences, graphs, trees). It allows the user to define in a flexible way by combining monotone primitives (e.g., syntactic, aggregate) a large set of constraints, the so-called primitive-based constraints. A preliminary version of this framework has been presented in [39,40], but it was restricted to the itemsets for transactional data. In Section 3, we show that this framework not only is a superclass of both kinds of monotone constraints and their boolean combinations but also other classes such as convertible and succinct constraints (a result about the scope of the primitive-based framework was given in [40] but it was limited to the itemsets and monotone constraints). The second contribution tackles the mining of constraint patterns. The main idea has been given above: instead of trying to directly push the constraint q itself given by the user, we propose to use a relaxed form of q to provide suitable pruning conditions. The great interest of this approach is that we are able to automatically infer monotone and anti-monotone properties which can be efficiently pushed by usual constraint mining algorithms, even if q does not satisfy such properties. For this purpose, we define and use two virtual patterns as a trick in order to automatically achieve the pruning conditions from any primitive-based constraint. These virtual patterns synthesize the search space by taking into account both the specificities of the data mining context and the constraint q. Furthermore, our approach enables us to simultaneously push these (anti-)monotone constraints, which is generally more efficient than handling them sequentially. We have introduced in [39] the idea of virtual patterns, but they were only linked to the data and defined for the itemsets. In this paper, we first generalize this idea to any pattern language and second, by considering also the specificities of q, we get more powerful pruning conditions. Furthermore, as a theoretical result, we show in which situations these pruning conditions are optimal.

The paper is organized in the following way. Section 2 introduces the motivations, the related work on constraint-based mining and the key idea of the relaxation. The primitive-based framework and its scope are given in Section 3. Section 4 provides the bounding operators which are necessary to relax the primitive-based constraints. The virtual patterns are defined in Section 5. Section 6 shows how the virtual patterns, combined with the bounding operators yield monotone relaxations and suitable pruning conditions. Moreover, we show in which cases the relaxation is optimal. Finally, Section 8 gives experimental results showing the effectiveness of our approach. Section 9 concludes.

2. Context and motivations

2.1. Constraint-based pattern mining problem

The task of mining patterns under constraints has been formalized by the framework of Mannila and Toivonen [27]. Given a *database* $\mathbf{r}$, a *language* $\mathcal{L}$ gathering all the patterns, and a selection predicate q (called *constraint*) to evaluate the interestingness of a pattern, constraint-based mining aims at extracting all the patterns of $\mathcal{L}$ satisfying the constraint q in $\mathbf{r}$. The set of mined patterns, denoted by $\mathcal{T}h(\mathcal{L}, \mathbf{r}, q)$, is called *theory*. The result of the constraint q for a given pattern φ generally depends on the database $\mathbf{r}$ (e.g., the minimal frequency constraint). So, we should write $q(\varphi, \mathbf{r})$. To alleviate the notations, unless otherwise indicated, $q(\varphi)$ refers to $q(\varphi, \mathbf{r})$.

Let us provide a few examples covering several pattern languages. We start with the problem of itemset mining originally introduced so as to derivate association rules [1].

Example 1 Itemset mining. *Given a set of items $\mathcal{I}$, the itemset language $\mathcal{L}_\mathcal{I}$ is the powerset of items $2^\mathcal{I}$ and a dataset is a multi-set of patterns in $\mathcal{L}_\mathcal{I}$. Each database entry is called a transaction. For instance, $\mathcal{I} = \{A, B, C, D, E, F\}$ and $\mathcal{D} = \{\{A, B, E, F\}, \{A, E\}, \{A, B, C, D\}, \{A, B, C, D, E\}, \{D, E\}, \{C, F\}\}$. The aim is to extract all the itemsets of $\mathcal{L}_\mathcal{I}$ occurring in $\mathcal{D}$ and satisfying a given constraint. The minimal frequency constraint (i.e., patterns occurring in $\mathbf{r}$ more than a fixed threshold) is likely the most usual constraint.*

The empty set is usually removed from $\mathcal{L}_\mathcal{I}$ and an itemset is denoted by a string notation (e.g., ABC denotes $\{A, B, C\}$).

The next example addresses sequential data. Sequential pattern mining has been introduced in [3]. It is useful whenever database entries depend on an order.

Example 2 Sequence mining. *Given a set of items $\mathcal{I}$, a sequence is an ordered multi-set of itemsets. Thereby, the language of sequences $\mathcal{L}_\mathcal{S}$ corresponds to the set of the all possible sequences and a dataset is a multi-set of sequences belonging to $\mathcal{L}_\mathcal{S}$. For instance, $\mathcal{I} = \{A, B, C, D, E, F\}$ and $\mathcal{D} = \{\langle(C)(A)\rangle, \langle(AB)(C)(ADF)\rangle, \langle(ACE)\rangle, \langle(C)(AD)(A)\rangle, \langle(B)\rangle\}$. The sequence mining task aims at discovering all the sequences present in $\mathcal{D}$ and satisfying a given constraint. Again, the minimal frequency constraint (i.e., sequences occurring in $\mathcal{D}$ bigger than a fixed threshold) is very usual.*

The language $\mathcal{L}$ may be infinite as in Example 2 where an item can be indefinitely repeated. There is also no limitation about the database $\mathbf{r}$ and no particular relation between $\mathcal{L}$ and $\mathbf{r}$ even if in practice a database is often composed of a multi-set of patterns of $\mathcal{L}$ (see Examples 1 and 2). Additional tables of values are required for some constraints as the average constraint, as shown in Example 3. Such constraints are very useful for guiding the discovering task.

Example 3 Average constraint. *Given a function $val : \mathcal{I} \rightarrow \Re^+$ (see below the table of values), we extend val to an itemset X and note $X.val$ the multi-set $\{val(a) \mid a \in X\}$. For a sequence, we append together the multi-sets corresponding to each individual itemset. This kind of function is used with the usual* SQL*-like primitives: sum, min, avg and so on.*

Item	A	B	C	D	E	F
val	50	30	75	10	30	15

Thereby, the average constraint corresponds to $avg(X.val) \geqslant \alpha$ *where* $avg(X.val)$ *is the average value of the multi-set* $X.val$ *and* α *is the minimal average threshold. For instance, the itemset* AB *whose average equals 40* $(= (50 + 30)/2)$*, satisfies the constraint* $avg(X.val) \geqslant 40$*.*

Finally, let us say that constraint-based mining addresses many other patterns, generally more complex, such as graphs [37], molecules [25] or strings [16].

2.2. Monotonicity

Mining patterns under constraints requires the exploration of the search space depicted by $\mathcal{L}$. Unfortunately, this space is generally huge and it is necessary to use pruning conditions. A lot of pruning conditions are based on the property of monotonicity. This property requires a partial order on $\mathcal{L}$ to structure it and we need to recall the specialization/generalization relation proposed by Mitchell [29] in the field of concept-learning. A *specialization relation* is a partial order $\preceq$ on the patterns in $\mathcal{L}$. φ is said to be *more general* (resp. *more specific*) than γ, iff $\varphi \preceq \gamma$ (resp. $\gamma \preceq \varphi$). When $\varphi \preceq \gamma$ and $\varphi \neq \gamma$, φ is said strictly more general than γ and we write $\varphi \prec \gamma$. For instance, the set inclusion $\subseteq$ is a specialization relation (e.g., A is more general than AB and we have $A \subseteq AB$). Similarly, a sequence $X = \langle x_1 x_2 \dots x_n \rangle$ is more general than another $Y = \langle y_1 y_2 \dots y_m \rangle$ (denoted by $X \preceq_{\mathcal{S}} Y$ where $\preceq_{\mathcal{S}}$ is the specialization relation on $\mathcal{L}_{\mathcal{S}}$) if there exist integers $i_1 < i_2 < \ldots < i_n$ such as $x_1 \subseteq y_{i_1}$, $x_2 \subseteq y_{i_2}, \ldots, x_n \subseteq y_{i_n}$.

A constraint q is monotone with respect to (w.r.t.) the specialization $\preceq$ iff whenever $\gamma \preceq \varphi$ and $q(\gamma, \mathbf{r}) = true$, we have $q(\varphi, \mathbf{r}) = true$. Dually, a constraint q is anti-monotone w.r.t. $\preceq$ iff whenever $\gamma \preceq \varphi$ and $q(\varphi, \mathbf{r}) = true$, we have $q(\gamma, \mathbf{r}) = true$. For instance, the constraint of minimal frequency (introduced in Examples 1 and 2) is anti-monotone w.r.t $\preceq$. We give now a more formal definition of this constraint.

The frequency of a pattern X (denoted $freq(X)$) is the number of transactions in $\mathcal{D}$ containing X w.r.t. the specialization relation. For instance, $freq(AB) = 3$ (see Example 1) and $freq(\langle (AB) \rangle) = 1$ (see Example 2). Let γ be the minimal frequency threshold, $freq(X) \geqslant \gamma$ designates the minimal frequency constraint. Continuing Example 1, the itemset AB satisfies the constraint $freq(X) \geqslant 3$ because AB occurs in 3 transactions in $\mathcal{D}$. On the contrary, the itemset F does not satisfy this constraint (its frequency is only 2). As the minimal frequency is anti-monotone and AB satisfies it with $\gamma = 3$, the generalizations of AB (i.e., A and B) also satisfy this constraint.

The next section indicates the main methods for constraint-based mining and points out the great role of the monotonicity during the mining step.

2.3. Methods for constraint-based mining

Most constraint-based mining algorithms take advantage of monotonicity which offers pruning conditions to safely discard patterns from the search space. Indeed, when a pattern X does not satisfy a monotone (resp. anti-monotone) constraint, any generalization (resp. specialization) of X does not satisfy the constraint either. Several frameworks exploit this principle to mine a monotone or an anti-monotone constraint.

The *levelwise algorithm* [27] generalizes the generate-and-test approach originally proposed by the APRIORI algorithm [2]. Another generic algorithm is presented in [47] with sharp details about implementation of general data structures. The *inductive databases* framework [21] combines several constraints having suitable properties of monotonicity to build more complex constraints. In particular,

a randomized method [14] mines a conjunction of one monotone and one anti-monotone constraints whose solution space corresponds to a version space in the field of machine learning [29]. For example, the mining of emerging patterns can be expressed as a disjunction of version spaces [11]. An algebra is proposed to evaluate and optimize such inductive queries [26]. In [18], the answers are completed by a measure function (by considering its monotonicity).

More efficient algorithms have been designed to specific languages or classes of constraints. For the itemset language, several methods have been proposed to extract a conjunction of one monotone constraint and one anti-monotone constraint. They are based on direct approaches [8,10,13] or pre-processing techniques [7]. They benefit from specific data structures (e.g., trie [24], FP-tree [20], COFI-tree [12]) or properties as the dualization [19]. Some other approaches are devoted to specific kinds of patterns such as the "closed patterns" [32,34]. Nevertheless, such tricks are difficult to generalize for any language. For instance, even if the notion of "closed pattern" is extended to few popular languages (e.g., sequences [43] or trees [42]), the corresponding closure operator cannot be trivially defined for any language.

Several classes of constraints cover a larger set of queries than the previous ones. Figure 4 depicts the *succinct* [31], *convertible* [33] and *loose anti-monotone* [9] classes of constraints. But, these classes are confined to the itemset language (except for the convertible one which is extended to sequences [36]) and their algorithms are incompatible with another language. Moreover, in some cases, like the class of convertible constraints, the algorithm needs a total order over itemsets to mine the patterns. For instance, the minimal average constraint (see Example 3) is convertible anti-monotone with $C < A < B < E < F < D$. Given an order relation R over items, whenever an itemset X satisfies a convertible anti-monotone constraint, any prefix of X w.r.t. R also satisfies this constraint. On our example, as the itemset AB satisfies $avg(X.val) \geqslant 40$, we are sure that its prefix A also satisfies the constraint. Let us note that B is not a prefix of AB. Convertibility exploits an order relation through its specialization relation (based on prefixes), but it is basically similar to monotonicity.

As we have seen above, the most generic frameworks in term of languages are restricted to constraints satisfying a monotone property. In fact, these approaches are pruning oriented: they are efficient to reduce the search space. But, they strongly reduce the expressiveness of the user and its expectancies are not satisfied [31]. At the same time, mining algorithms dedicated to large classes of user-specified constraints are only limited to several particular languages. To the best of our knowledge, there is no method to process the pattern mining task for any language with a constraint which is not a boolean formulae of monotone constraints. This paper copes with this problem using the *relaxation of constraints*.

2.4. Problem statement and constraint relaxation

A lot of useful constraints such as the minimal area constraint (cf. Example 4) do not deal with the classes of constraints presented above and are very tough. Let X be an itemset and $count(X)$ its number of items (or its cardinality). Similarly, the length of a sequence is the sum of the lengths of its itemsets. For instance, $count(AB) = 2$ and $count(\langle(AB)(A)\rangle) = 3$.

Example 4 Minimal area constraint. *The minimal area constraint is defined as* $freq(X) \times count(X) \geqslant \rho$ *where* ρ *is the minimal area threshold.* $area(X)$ *directly refers to* $freq(X) \times count(X)$.

The minimal area constraint does not satisfy monotonicity properties: it is neither monotone ($area(ABC) \geqslant 6$ but $area(ABCDE) < 6$), nor anti-monotone ($area(BC) < 6$ but $area(ABC) \geqslant 6$).

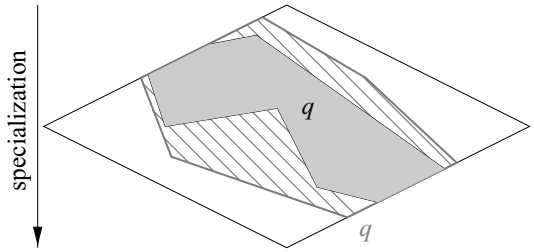


Fig. 1. A relaxation q' of constraint q.

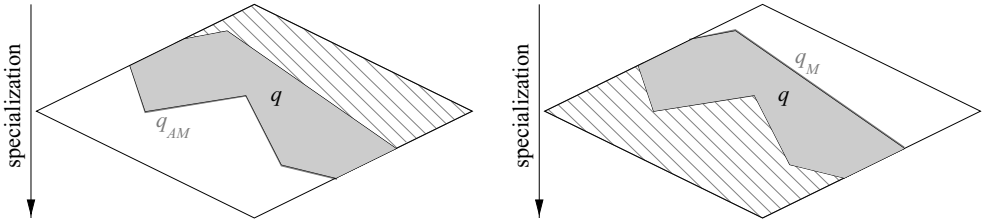


Fig. 2. Monotone and anti-monotone relaxations of a constraint q.

We will see that the minimal area constraint and many others (see Example 5) belong to the primitive-based framework (cf. Section 3.1) and consequently can be efficiently mined thanks to our framework. Let us note that many works extract closed patterns [32] (also named tiles [17] or blocks [15]). Closed patterns are a subset of the patterns satisfying the minimal area constraint and mining the closed patterns does not provide the complete collection of patterns satisfying the minimal area constraint. In our approach, we preserve the completeness for the minimal area constraint and more generally for any constraint.

A "naive" idea to mine constraint patterns is to enumerate all the patterns of any language $\mathcal{L}$ occurring at least once in $\mathbf{r}$ (with a levelwise algorithm) and to test whether they satisfy or not the desired constraint. This approach naturally fails whenever the considered dataset contains an overwhelming number of patterns. In this paper, we suggest to approximate the theory of the original constraint q by a larger collection corresponding to the solution space of a slightly weaker constraint q': $\mathcal{T}h(\mathcal{L}, \mathbf{r}, q) \subseteq \mathcal{T}h(\mathcal{L}, \mathbf{r}, q')$. The less restrictive constraint q' induced by q, is called a *relaxation*, and we have the following implication $q \Rightarrow q'$. Figure 1 depicts the theory of q (i.e., the gray shape) and its relaxation q' (i.e., the hatched shape).

The key idea is to get a relaxed constraint having suitable monotonicity properties in order to re-use usual algorithms as the levelwise one. More precisely, given a language $\mathcal{L}$, a database $\mathbf{r}$ and a constraint q, we want to automatically find one monotone relaxation and one anti-monotone relaxation of q as there are efficient algorithms for such a pair of constraints (see Section 2.3). Figure 2 depicts the desired anti-monotone q_{AM} and monotone q_M relaxations of q. Then a simple filter selects the patterns satisfying q. Such an approach is *discovery preserving* [5] since the pruning coming from the relaxation does not remove patterns satisfying q.

In literature, relaxation is used to discover more unexpected patterns [4] or introduce softness in order to avoid the crisp effect of the boolean selection [6]. This concept is also artfully exploited to extract constrained itemsets in wide datasets by means of transposition [22]. In our context, the relaxation is a technical method to make feasible extractions under complex constraints and can be seen as a kind of pre-processing of the user-specified constraint.

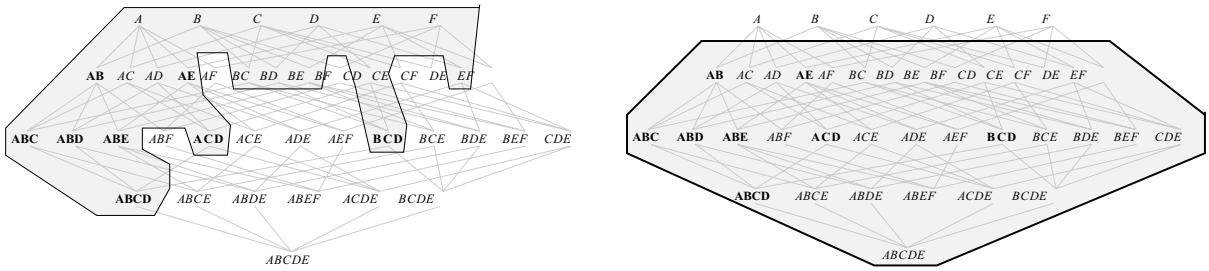


Fig. 3. An anti-monotone relaxation (on left) and a monotone relaxation (on right) of the constraint $area(X) \geqslant 6$ (itemsets in bold satisfy this constraint).

The non-trivial problem of relaxation is partially solved in specific cases. Regular expression constraints are relaxed into anti-monotone relaxations for mining significant sequences [16]. Two constraints based on chi-squared and correlation are relaxed to find itemsets [30]. In the field of itemsets, a collection of boolean formulae of monotone constraints [40] and aggregate constraints [44] can also be relaxed. Compared to these approaches, this paper provides a more general answer to the relaxing problem by focusing on the primitive-based constraints for any language.

Figure 3 illustrates the effectiveness of the relaxation by coming back to our example of the minimal area constraint. Assuming that we are interested in all the itemsets present in the dataset $\mathcal{D}$ (see Example 1) and satisfying the minimal area constraint with the threshold $\rho = 6$ (bold itemsets). The naive approach requires to mine $\mathcal{T}h(\mathcal{L}_{\mathcal{I}}, \mathbf{r}, freq(X) \geqslant 1)$ corresponding to 40 itemsets. The anti-monotone constraint $freq(X) \geqslant 2$ and the monotone constraint $count(X) \geqslant 2$ are relaxations of $area(X) \geqslant 6$. These relaxations lead to three different mining strategies: $\mathcal{T}h(\mathcal{L}_{\mathcal{I}}, \mathbf{r}, freq(X) \geqslant 2)$ (21 itemsets, on left), $\mathcal{T}h(\mathcal{L}_{\mathcal{I}}, \mathbf{r}, count(X) \geqslant 2)$ (34 itemsets, on right) and $\mathcal{T}h(\mathcal{L}_{\mathcal{I}}, \mathbf{r}, freq(X) \geqslant 2 \wedge count(X) \geqslant 2)$ (15 itemsets). The theory of $area(X) \geqslant 6$ has 8 itemsets: AB, AE, ABC, ABD, ABE, ACD, BCD, and $ABCD$. We will give in Section 6.1 the relaxations of the minimal area constraint achieved thanks to the primitive-based framework. These relaxations lead to examine only the 15 itemsets corresponding to $\mathcal{T}h(\mathcal{L}_{\mathcal{I}}, \mathbf{r}, freq(X) \geqslant 2 \wedge count(X) \geqslant 2)$.

3. The primitive-based framework

This section first defines the primitive-based framework for any partially ordered language (i.e., a language with its specialization relation). Then we show that the primitive-based framework defines a superset of the usual classes of constraints, allowing us to define numerous and varied constraints.

3.1. The primitive-based constraints

We call "primitive-based constraints" the set of constraints defined by the primitive-based framework. Contrary to the usual classes of constraints (e.g. monotone, convertible) the primitive-based constraints are based on a set of primitives as defined in Definition 1:

Definition 1 Primitive. *A primitive p is a monotone function according to any variable when the others remain constant.*

There is no restriction about the arity of the primitives. But, for a given primitive, Definition 1 implies that the domain of each variable and its range have to be partially (or totally) ordered sets (e.g., the

language $\mathcal{L}$, $\Re^+$). The set of primitives is denoted by $\mathcal{P}$. For instance, $freq$ and $count$ (cf. Example 4) are primitives of our framework because they are respectively a decreasing and an increasing function w.r.t. their variable. Thus they are monotone functions.

Many primitives evaluate a pattern of the language $\mathcal{L}$ (e.g., $freq$ or $count$) and these primitives are said *terminal*. The other primitives (e.g., $\times$ or $\geqslant$) are useful to combine the previous ones (see Examples 3 and 4). Even if the evaluation of a primitive p often depends on the database $\mathbf{r}$, we omitted $\mathbf{r}$ by writing $p(\varphi)$ instead of $p(\varphi, \mathbf{r})$. A primitive may also have different meanings according to $\mathcal{L}$. For instance, in Example 4, $count$ has a different meaning according to the itemset language $\mathcal{L}_{\mathcal{I}}$ or the sequence language $\mathcal{L}_{\mathcal{S}}$.

In practice, more complex primitives are useful to the user. For instance, the area function is not monotone, but it is a combination of several primitives of $\mathcal{P}$: the $area$ is decomposed into $freq(X) \times count(X)$. This kind of combination can be seen as a high-level primitive:

Definition 2 High-level primitive. *A high-level primitive h defined over language $\mathcal{L}$ is a recursive combination of several primitives in $\mathcal{P}$.*

The set $\mathcal{H}_{\mathcal{L}}$ denotes all the high-level primitives on $\mathcal{L}$. For a given high-level primitive h, the highest number of combinations is called the *degree* and noted $\deg h$. More precisely, the high-level primitives of degree 0 put together all the unary functions defined over $\mathcal{L}$ (i.e., the terminal primitives). Besides, each high-level primitive h of degree n is compounded of one primitive p with k high-level primitives h_i of inferior degree such that we have $h = p(h_1, \ldots, h_k)$. The form $p(h_1, \ldots, h_k)$ is called the decomposition of h. The degree of at least one high-level primitive h_i has to equal $n - 1$ in order to ensure that any expression has a unique decomposition. To sum up, whenever a high-level primitive h is a terminal one, its degree is 0 and otherwise, its degree corresponds to $1 + \max_{i \in \{1,\ldots,k\}} \deg h_i$ where $p(h_1, \ldots, h_k)$ is the decomposition of h. For instance, as $freq$ and $count$ are monotone primitives from $\mathcal{L}_{\mathcal{I}}$ or $\mathcal{L}_{\mathcal{S}}$ to $\Re^+$, their degree is 0. Thus, the degree of $area$ is equal to 1 (i.e., $\deg area = 1$) because its prefix decomposition is $\times(freq, count)$ and $\mathcal{P}$ contains $\times$.

Finally, a *primitive-based constraint* (PBC in summary) is a constraint which is a high-level primitive defined from $\mathcal{L}$ to $\mathcal{B} = \{true, false\}$:

Definition 3 Primitive-based constraint. *A primitive-based constraint is a high-level primitive defined from language $\mathcal{L}$ to booleans $\mathcal{B}$.*

The set of such constraints defined over $\mathcal{L}$ is denoted by $\mathcal{Q}_{\mathcal{L}}$. Following on, if the specified language is clear, $\mathcal{Q}$ refers to $\mathcal{Q}_{\mathcal{L}}$. Table 1 provides an example of a set of constraints recursively defined corresponding to the particular primitives $\{\wedge, \vee, \neg, <, \leq, \subset, \subseteq, +, -, \times, /, freq, count, sum, max, min, \cup, \cap, \setminus\}$.

Let $freq(X, \mathcal{D}_i)$ be the frequency of pattern X on the sub-dataset $\mathcal{D}_i \subseteq \mathcal{D}$. Example 5 provides examples of primitive-based constraints belonging to $\mathcal{Q}_{\mathcal{L}_{\mathcal{I}}}$ or $\mathcal{Q}_{\mathcal{L}_{\mathcal{S}}}$.

Example 5 Primitive-based constraints. *Examples of PBC:*
$q_1(X) \equiv freq(X) \times count(X) \geqslant \gamma$
$q_2(X) \equiv (\min X + \max X)/2 \leqslant 50$
$q_3(X) \equiv sumX/count(X) \geqslant 25$
$q_4(X) \equiv AE \subseteq X$
$q_5(X) \equiv freq(X) \geqslant 2$

Table 1
A subset of the primitive-based constraints $\mathcal{Q}_{\mathcal{L}_\mathcal{I}}$

Constraint $q \in \mathcal{Q}_{\mathcal{L}_\mathcal{I}}$	Primitive(s)	Operand(s)
$q_1 \theta q_2$	$\theta \in \{\wedge, \vee\}$	$(q_1, q_2) \in \mathcal{Q}^2_{\mathcal{L}_\mathcal{I}}$
θq_1	$\theta \in \{\neg\}$	$q_1 \in \mathcal{Q}_{\mathcal{L}_\mathcal{I}}$
$e_1 \theta e_2$	$\theta \in \{<, \leq\}$	$(e_1, e_2) \in \varepsilon^2$
$s_1 \theta s_2$	$\theta \in \{\subset, \subseteq\}$	$(s_1, s_2) \in \mathcal{S}^2$
constant $b \in \mathcal{B}$	–	–
Aggregate expression $e \in \varepsilon$	Primitive(s)	Operand(s)
$e_1 \theta e_2$	$\theta \in \{+, -, \times, /\}$	$(e_1, e_2) \in \varepsilon^2$
$\theta(s)$	$\theta \in \{freq, count\}$	$s \in \mathcal{S}$
$\theta(s.val)$	$\theta \in \{sum, \max, \min\}$	$s \in \mathcal{S}$
constant $r \in \Re^+$	–	–
Syntactic expression $s \in \mathcal{S}$	Primitive(s)	Operand(s)
$s_1 \theta s_2$	$\theta \in \{\cup, \cap, \setminus\}$	$(s_1, s_2) \in \mathcal{S}^2$
variable $X \in \mathcal{L}_\mathcal{I}$	–	–
constant $l \in \mathcal{L}_\mathcal{I}$	–	–

$q_6(X) \equiv freq(X)/count(X) \geqslant \rho$
$q_7(X) \equiv 2 \times count(X) - count(X) \leqslant \rho$
$q_8(X) \equiv (q_5(X) \vee q_6(X)) \wedge q_4(X)$
$q_9(X) \equiv (|\mathcal{D}_2|/|\mathcal{D}_1|) \times freq(X, \mathcal{D}_1)/freq(X, \mathcal{D}_2) \geqslant \rho$

The monotone primitives are of a great interest. Most of the desired primitives are monotone functions and this property does not appear in practice as an important restriction. Besides, a non-monotone primitive can often be split into several monotone ones [44].

3.2. Varied and meaningful constraints

This section gives some insights into the primitive-based framework and shows how the expectations of end-users are fulfilled. A key point is the flexibility which is offered to express a large and rich variety of constraints.

Usual approaches are based on a dictionary of constraints. Grammars are often simple enumerations of atomical constraints [9,31]. Such lists of predefined constraints reduce expressiveness and then, the interest of the constraint-based mining. On the contrary, the primitive-based constraints are based on a dictionary of primitives which is combinatorially richer than a dictionary of constraints. Solvers performing the primitive-based framework implement an infinity of varied atomical constraints which can be easily specified.

Moreover, even if each primitive has a simple semantic, their combination enables us to express constraints having a rich global semantic. On the one hand, the user accurately specifies which type of patterns is wished (e.g., exceptions, regularities, contrasts between several classes). A type is often followed by an *aggregate constraint* based on SQL-like primitives (e.g., $freq$, sum, max). On the other hand, *syntactical constraints* (e.g., size and content of patterns) is a way to interpret background knowledge into the constraint in order to eliminate trivial or inconsistent patterns. The primitive-based constraints allow the user to use both syntactical and aggregate constraints simultaneously. In practice, most of the special constraints depicted by the literature are primitive-based constraints: constraint revealing emerging patterns q_9 [11], item constraint q_4 [41], etc.

3.3. A general class of constraints

In this section, we compare the primitive-based constraints w.r.t. the current classes of constraints. We start by linking the notion of monotonicity to primitive-based constraints:

Property 1 Superclass of monotone constraints. *A monotone* (*or anti-monotone*) *constraint is also a primitive-based one.*

Proof. Let q be a monotone constraint according to the specialization, we have $\forall \varphi \preceq \gamma \wedge q(\varphi) \Rightarrow q(\gamma)$. With respect to $false < true$, we obtain that $q(\varphi) \leqslant q(\gamma)$ for all $\varphi \preceq \gamma$ i.e. q is an increasing function defined on booleans. Thus, q is a monotone primitive for our framework. We obtain the same result for the anti-monotone constraints. □

Therefore, the class of primitive-based constraints is more general than the monotone constraints and the anti-monotone constraints. Actually, several primitive-based constraints described in Example 5 are anti-monotone (q_5, q_6 and q_7) or monotone (resp. q_4).

The next property even proves that different atomical primitive-based constraints can be combined to get a more sophisticated one:

Property 2 Closure by boolean operators. *All the boolean combinations of primitive-based constraints are primitive-based constraints.*

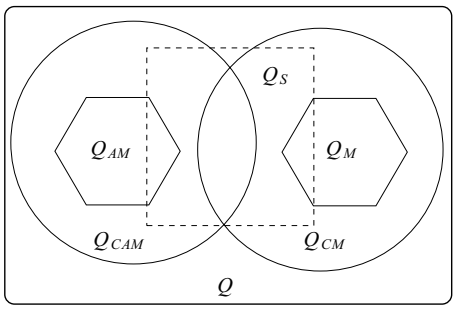
Proof. Let q_1 and q_2 be two primitive-based constraints. As we have $\{\wedge, \vee, \neg\} \subseteq \mathcal{P}$, we can combine the boolean operators with the constraints q_1 or/and q_2 by definition of primitive-based constraints. Thus, $q_1 \wedge q_2$, $q_1 \vee q_2$ and $\neg q_1$ are primitive-based constraints. □

In other words, Property 2 says that the primitive-based constraints constitute a Bool algebra. It means that in practice a user can improve a query by adding other atomical primitive-based constraints. Then, he focuses on few interesting patterns by means of highly selective multi-criteria.

The conjunction of Property 1 and Property 2 expresses that the primitive-based constraints constitute a larger class than inductive queries composed from both monotone and anti-monotone constraints. Thus, from our point of view, the primitive-based framework encompasses the abilities of the levelwise and inductive frameworks. Typically, the inductive query q_8 (see Example 5) belongs to our framework.

Although the monotone and anti-monotone constraints are only the general classes for any language $\mathcal{L}$, specific comparisons hold for the classes dedicated for specific languages. At first, a convertible constraint [33] is also a primitive-based constraint because it is only a monotone constraint by considering the language $\mathcal{L}_{\mathcal{I}}$ and the specialization relation based on prefixes (see Section 2.3). Secondly, any succinct constraint (which theory can be expressed in terms of strict powersets of sets of items using union and minus [31]) is always expressible in terms of monotone primitives. Then, any succinct constraint is a primitive-based constraint. Figure 4 summarizes these comparisons. The comparison of the primitive-based constraints with loose anti-monotone [9] or separable constraints [44] is still an open issue.

In addition to the rich semantic brought by the primitive-based constraints, Properties 1 and 2 show that these constraints are a superclass of all the combinations of both monotone and anti-monotone constraints. The relaxing approach proposed in the next sections is very general and it can be applied to numerous and varied constraints. Thereby, we can handle inductive queries and more complex constraints to deduce monotone and anti-monotone relaxations.



Q_M : monotone constraints
Q_{AM} : anti-monotone constraints
Q_S : succinct constraints
Q_{CM} : convertible monotone constraints
Q_{CAM} : convertible anti-monotone constraints
Q : primitive-based constraints

Fig. 4. Comparison between the classes of constraints (for itemset language $\mathcal{L}_{\mathcal{I}}$).

4. Handling the primitive-based constraints

After depicting the paradigm which is at the core of our framework, this section offers the operators enabling us to achieve monotone relaxed constraints.

4.1. Principle

As indicated above, primitive-based constraints enable the end-user to specify unusual constraints or unimaginable ones for the computer scientist. Very new and unexpected nuggets of knowledge may be extracted. Unfortunately, defining varied constraints and relaxing them are antagonist aims. Starting from singular and varied elements, we are looking for one general method which can be automated to reach these two aims.

Our framework handles primitives instead of handling constraints. The key idea is to take advantage of the local properties of each primitive in order to build global properties on constraints. This principle is performed in many tasks (e.g., itemset mining [38], detecting monotone constraints [40], etc.). In our context, we decline the same approach to automatically relax constraints. We mainly focus on the monotonicity of the primitives $\mathcal{P}$ to deduce sufficient conditions about the monotonicity of the high-level primitives $\mathcal{H}$ (including constraints $\mathcal{Q}$). Formally, we introduce two recursive operators to achieve and analyze relaxations.

Our method has several assets. First, it can be used with any language. As mentioned above, the primitives deeply depends on the language $\mathcal{L}$ and the database $\mathbf{r}$. But their manipulations are independent from these parameters. Second, the implementation of this paradigm is often natural. Briefly, the *constraint handlers* implementing each formal operator are based on two key points: (1) disposing of a dictionary of primitives with their own local properties and (2) recursively combining them to make a global property on the constraint.

4.2. The lower and upper bounding operators

This section gives the key operators which are necessary to relax primitive-based constraints for any language $\mathcal{L}$.

The interval between the patterns φ and γ (denoted $[\varphi, \gamma]$) corresponds to the set $\{\theta \in \mathcal{L} | \varphi \subseteq \theta \subseteq \gamma\}$. Given such an interval, we succeed to bound the constraint by using the principle described above: we bound primitives on this interval by observing their monotonicity and then, for any high-level primitive, we combine these local bounds in order to find a global one.

We start by giving the definition of the bounding operators denoted $\lfloor . \rfloor$ and $\lceil . \rceil$ [1]:

Definition 4 Bounding operators. *Let h be a high-level primitive and $[\varphi, \gamma]$ be an interval, $\lfloor h \rfloor \langle \varphi, \gamma \rangle$ and $\lceil h \rceil \langle \varphi, \gamma \rangle$ are defined as below:*

- *if $\deg h = 0$: $\lfloor h \rfloor \langle \varphi, \gamma \rangle = h(\varphi)$ and $\lceil h \rceil \langle \varphi, \gamma \rangle = h(\gamma)$ iff h is an increasing function. Otherwise h decreases, $\lfloor h \rfloor \langle \varphi, \gamma \rangle = h(\gamma)$ and $\lceil h \rceil \langle \varphi, \gamma \rangle = h(\varphi)$.*
- *if $\deg h \geqslant 1$: $\lfloor h \rfloor \langle \varphi, \gamma \rangle = p(h'_1, \ldots, h'_k)$ and $\lceil h \rceil \langle \varphi, \gamma \rangle = p(H'_1, \ldots, H'_k)$ where $p(h_1, \ldots, h_k)$ is the decomposition of h and for each variable $i \in \{1, \ldots, k\}$:*

$$\begin{cases} h'_i = \lfloor h_i \rfloor \langle \varphi, \gamma \rangle \text{ and } H'_i = \lceil h_i \rceil \langle \varphi, \gamma \rangle & \text{if } p \text{ increases with} \\ & \quad \text{the } i^{th} \text{ variable} \\ h'_i = \lceil h_i \rceil \langle \varphi, \gamma \rangle \text{ and } H'_i = \lfloor h_i \rfloor \langle \varphi, \gamma \rangle & \text{otherwise} \end{cases}$$

Property 3 will show that $\lfloor q \rfloor \langle \varphi, \gamma \rangle$ (resp. $\lceil q \rceil \langle \varphi, \gamma \rangle$) is a lower bound (resp. an upper bound) of the interval $[\varphi, \gamma]$ for q. In other words, these operators enables us to automatically compute the lower and upper bounds of $[\varphi, \gamma]$ for q. This result stems from the properties of increasing and decreasing of the primitives. Example 6 illustrates $\lfloor . \rfloor$ and $\lceil . \rceil$ on the minimal area constraint.

Example 6 Bounding the minimal area constraint. *As $\geqslant$ increases in $\mathcal{B}$ according to the first variable and decreases according to the second one, we have $\lfloor area(X) \geqslant 6 \rfloor \langle X, Y \rangle = \lfloor area(X) \rfloor \langle X, Y \rangle \geqslant \lceil 6 \rceil \langle X, Y \rangle$. As 6 is a constant and $\times$ increases with each variable, we obtain respectively that $\lceil 6 \rceil \langle X, Y \rangle = 6$ and $\lfloor area(X) \rfloor \langle X, Y \rangle = \lfloor freq(X) \rfloor \langle X, Y \rangle \times \lfloor count(X) \rfloor \langle X, Y \rangle$. Finally, $\lfloor area(X) \geqslant 6 \rfloor \langle X, Y \rangle$ is equal to $freq(Y) \times count(X) \geqslant 6$ because $freq$ decreases and $count$ increases. In the same way, $\lceil area(X) \geqslant 6 \rceil \langle X, Y \rangle$ is equal to $freq(X) \times count(Y) \geqslant 6$.*

Starting from h and $[\varphi, \gamma]$, the bounding operators are recursively applied and lead to automatically compute a lower and an upper bounds of $[\varphi, \gamma]$ for h. Property 3 is a major result to get the relaxations in Section 6.

Property 3 Bounds of an interval. *$\lfloor q \rfloor$ and $\lceil q \rceil$ are respectively a lower bound and an upper bound of primitive-based constraint q. Given an interval $[\varphi, \gamma]$ and a pattern $\theta \in [\varphi, \gamma]$, we have $\lfloor q \rfloor \langle \varphi, \gamma \rangle \leqslant q(\theta) \leqslant \lceil q \rceil \langle \varphi, \gamma \rangle$.*

We start by giving Lemma 1 which facilitates the understanding of this property. This lemma expresses that the accuracy of the bounding operators increases when the size of the interval decreases.

Lemma 1 [39]. *Let $h \in \mathcal{H}$ and $[\varphi_1, \gamma_1] \subseteq [\varphi_2, \gamma_2]$, we have $\lfloor h \rfloor \langle \varphi_1, \gamma_1 \rangle \geqslant \lfloor h \rfloor \langle \varphi_2, \gamma_2 \rangle$ and $\lceil h \rceil \langle \varphi_1, \gamma_1 \rangle \leqslant \lceil h \rceil \langle \varphi_2, \gamma_2 \rangle$.*

According to the language gathering all the intervals with the specialization relation $\subseteq$, Lemma 1 means that the lower and the upper bounding operators are respectively anti-monotone and monotone. We can now prove Property 3:

Proof. Let q be a primitive-based constraint and θ be a pattern of $[\varphi, \gamma]$. As we have $[\theta, \theta] \subseteq [\varphi, \gamma]$, Lemma 1 gives that $\lfloor h \rfloor \langle \theta, \theta \rangle \geqslant \lfloor h \rfloor \langle \varphi, \gamma \rangle$ and $\lceil h \rceil \langle \theta, \theta \rangle \leqslant \lceil h \rceil \langle \varphi, \gamma \rangle$. Obviously, $\lfloor h \rfloor \langle \theta, \theta \rangle = \lceil h \rceil \langle \theta, \theta \rangle =$

[1] To alleviate the notations, we replace $\lfloor . \rfloor([\varphi, \gamma])$ by $\lfloor . \rfloor \langle \varphi, \gamma \rangle$.

$q(\theta)$ and then, we deduce that $\lfloor h \rfloor \langle \varphi, \gamma \rangle \leqslant h(\theta) \leqslant \lceil h \rceil \langle \varphi, \gamma \rangle$. In the particular case of constraint, we conclude that Property 3 is right. □

Property 3 justifies the name of the bounding operators $\lfloor . \rfloor$ and $\lceil . \rceil$. These bounds of a constraint are effective in practice thanks to the monotone primitives. First, we claim that the bounds are quite accurate. In the particular case of (anti-)monotone constraints, these bounds are even exact (see Section 7.1). Second, the two functions $\lfloor q \rfloor$ and $\lceil q \rceil$ (which map an interval to a boolean) do not vary with the database or the interval. In other words, practical utilization of these functions requires only one computation before the beginning of the pattern extraction and then, has a very low computational cost. Third, as the boolean operators are monotone primitives, our framework allows us to directly use constraints containing conjunctions or disjunctions of other constraints.

These top-level operators are useful to obtain efficient pruning conditions of a primitive-based constraint on an interval [38]. Basically, if an upper bound of q on $[\varphi, \gamma]$ is equal to $false$, no pattern of $[\varphi, \gamma]$ satisfies q (because they are all less than $false$) and then, we can prune this interval. MUSIC [38] exploits this interval pruning in order to drastically reduce the search space for a primitive-based constraint.

Our pruning strategy (cf. Section 6) uses the bounding operators on intervals gathering all the generalizations or all the specializations of φ. Indeed, whenever any specialization of φ is less than $false$, we can safely prune φ. For instance, for the specific itemset language, $[X, \mathcal{I}]$ clearly describes the more specific itemsets than X. Unfortunately, the most specific pattern $\mathcal{I}$ (or analog one) does not exist for any language. Next section shows how to get appropriate intervals by introducing artificial patterns named virtual patterns.

5. The most general and specific virtual patterns of a version space

5.1. Definition of virtual patterns

This section introduces two artificial patterns which summarize a version space leading to efficient intervals for our relaxation strategy. These patterns are said virtual because they have unexpected properties.

Let us recall that a version space [29] is a convex collection of patterns. It corresponds exactly to the theory of the conjunction of one monotone and one anti-monotone constraints. In term of intervals, a set $VS \subseteq \mathcal{L}$ is a version space iff whenever $\varphi \in VS$ and $\gamma \in VS$ such that $\varphi \preceq \gamma$, we have $[\varphi, \gamma] \subseteq VS$. For instance, the collection of the patterns present at least once in a database is a version space (whenever this collection is finate). Indeed, if φ and γ are present in $\mathbf{r}$, all the patterns θ between φ and γ are also in $\mathbf{r}$. Thereafter, this collection, denoted by $\mathcal{C}$, is extensively used.

We start by giving the definition of the virtual patterns:

Definition 5 The most general and specific virtual patterns. *Let VS be a version space, the most general virtual pattern $\bot(VS)$ and the most specific virtual pattern $\top(VS)$ are defined as follow for each function $p : \mathcal{L} \to S$ (where S is a totally ordered set):*

$$p(\bot(VS)) = \begin{cases} \min_{\varphi \in VS} p(\varphi), & \text{if } p \text{ is an increasing function} \\ \max_{\varphi \in VS} p(\varphi), & \text{if } p \text{ is a decreasing function} \end{cases}$$

$$p(\top(VS)) = \begin{cases} \max_{\varphi \in VS} p(\varphi), & \text{if } p \text{ is an increasing function} \\ \min_{\varphi \in VS} p(\varphi), & \text{if } p \text{ is a decreasing function} \end{cases}$$

The virtual patterns synthesize the specificities of a version space (e.g., stemmed from $\mathbf{r}$) according to the different primitives. Indeed, any pattern of the version space has a value of a given primitive p between the values of the virtual patterns.

The virtual patterns $\bot(VS)$ and $\top(VS)$ only depend on the version space VS and the database $\mathbf{r}$. For instance, Example 7 presents the most and the least specific virtual itemsets of the patterns present in $\mathcal{D}$. The values of sum, min or max for each virtual pattern is linked to the table of values given in Example 3.

Example 7 Virtual itemsets of $\mathcal{C}$. *We consider here the most general and specific virtual itemsets of the collection $\mathcal{C}$ (i.e., $\mathcal{C} = \mathcal{T}h(\mathcal{L}_\mathcal{I}, \mathbf{r}, freq(X) \geqslant 1)$). Considering the database of Example 1 and Example 3, we have $count(\bot(\mathcal{C})) = \min_{X \in \mathcal{C}} count(X)$ because the count is an increasing function on $\mathcal{L}_\mathcal{I}$. As the size of the shortest patterns is equal to 1, we obtain that $count(\bot(\mathcal{C})) = 1$. With similar reasonings, we obtain the following table:*

Primitive p	$p(\bot(\mathcal{C}))$	$p(\top(\mathcal{C}))$
$freq$	*4*	*1*
$count$	*1*	*5*
sum	*10*	*185*
min	*75*	*10*
max	*10*	*75*
...	...	...

Following on, the closure of the version space VS, denoted $\overline{VS}$, corresponds to the version space VS completed with the most general and specific virtual patterns: $\overline{VS} = VS \cup \{\bot(VS), \top(VS)\}$. The specialization relation $\preceq$ is extended to $\overline{VS}$ by assuming that the most general virtual pattern $\bot(VS)$ is more general than any pattern of $\overline{VS}$. Dually, the most specific virtual pattern $\top(VS)$ is more specific than any pattern of $\overline{VS}$. Then, any pattern of the version space VS is included between $\bot(VS)$ and $\top(VS)$ (i.e., $VS \subseteq [\bot(VS), \top(VS)]$). We also have $\overline{VS} = [\bot(VS), \top(VS)]$. From Example 7, we show that the virtual patterns have unexpected properties:

Example 8 Unexpected properties of virtual itemsets. *$\bot(\mathcal{C})$ is more general than each item (e.g., $\bot(\mathcal{C}) \subseteq A$ and $\bot(\mathcal{C}) \subseteq B$). Thereby, $\bot(\mathcal{C})$ would be the empty set because $A \cap B = \emptyset$. This result contradicts that its length (i.e., its cardinality) is equal to 1 (see the second line in Example 7). In the same way, each transaction of the dataset $\mathcal{D}$ is included in $\top(\mathcal{C})$ i.e., $ABEF \subseteq \top(\mathcal{C})$, $AE \subseteq \top(\mathcal{C})$, $ABCD \subseteq \top(\mathcal{C})$, etc. So, $\top(\mathcal{C})$ would equal $ABCDEF$, but its length is only 5. Same observations hold for the sequences.*

Example 8 clearly highlights that the most general and specific virtual patterns have non-realistic behaviors. We will see in Section 6 that these unexpected properties will lead to better intervals for the pruning strategy based on the relaxation. [37] also introduced an artificial element $\top$ to find molecular fragments, but the latter has expected properties.

5.2. Properties of virtual patterns

In this section, we show that the computation of virtual patterns is efficient thanks to the borders of a version space. It is an important point because our relaxing strategy requires these patterns.

A version space can be represented by two *borders* (or boundaries). The set G of the most general patterns of the version space (i.e., $G(VS) = \{\varphi \in VS |$ there is no $\gamma \in VS$ such that $\gamma \prec \varphi\}$) and dually, the set of the most specific patterns of the version space (i.e., $S(VS) = \{\varphi \in VS |$ there is no $\gamma \in VS$ such that $\varphi \prec \gamma\}$). Together, the sets G and S delimit the version space. Therefore, any pattern of a version space is comprised between a pattern of G and another one of S.

As borders G and S sum up a version space, it is expected that the virtual patterns of a version space can be deduced from its borders. Property 4 established this link:

Property 4 Border summary. *The most general and specific virtual patterns of a version space VS are respectively equal to the most general virtual pattern of the border $G(VS)$ and the most specific virtual pattern of the border $S(VS)$.*

Proof. Let $p : \mathcal{L} \rightarrow S$ be an increasing function. For each pattern $\varphi \in VS$, there exists $\gamma \in G(VS)$ such that $\gamma \preceq \varphi$. As p increases, we obtain that $p(\gamma) \leqslant p(\varphi)$ and $\min_{\gamma \in G(VS)} p(\gamma) \leqslant \min_{\varphi \in VS} p(\varphi)$. As $G(VS) \subseteq VS$, we conclude that $\min_{\varphi \in VS} p(\varphi) = \min_{\gamma \in G(VS)} p(\gamma)$. The three other relations are proved with a similar reasoning. □

Property 4 expresses that we have $\bot(VS) = \bot(G(VS))$ and $\top(VS) = \top(S(VS))$. These relations highlight that the most general (resp. specific) virtual patterns summarizes the knowledge about the most general (resp. specific) patterns w.r.t. the relation $\preceq$. The virtual patterns form a more condensed representation of the version space than their respective borders, which is a helpful representation for our relaxing problem (see Section 6). Besides, this property ensures that the patterns which belong to the borders, are sufficient to compute the virtual patterns. In practice, instead of enumerating all the patterns of the version space VS to compute $\bot(VS)$ and $\top(VS)$, we only analyze the borders $G(VS)$ and $S(VS)$. Whenever the borders are moderately small, the computation of virtual patterns is efficient. For instance, computing $\bot(\mathcal{C})$ (resp. $\top(\mathcal{C})$) in Example 7 only requires to consider the different items (resp. transactions).

Now we study the particular case where a virtual pattern is "real":

Property 5. *If the border $G(VS)$ contains only one pattern φ, then the most general virtual pattern of version space VS is exactly φ. Similarly, if the border $S(VS)$ contains only one pattern φ, then the most specific virtual pattern of version space VS is exactly φ.*

Proof. Let $G(VS)$ be equal to $\{\varphi\}$. Let $p : \mathcal{L} \rightarrow S$ be an increasing function. As $p(\bot(VS))$ is equal to $\min_{\varphi \in G(VS)} p(\varphi)$, we obtain $p(\bot(VS)) = p(\varphi)$. The same result holds for a decreasing function p. Thereby, $\bot(VS) = \varphi$. The other relation is proved with a similar reasoning. □

In other words, Property 5 expresses that if a border of VS is reduced to one element, the corresponding virtual pattern really belongs to VS. In particular, as the borders $G(\overline{VS})$ and $S(\overline{VS})$ respectively correspond to $\{\bot(VS)\}$ and $\{\top(VS)\}$, the virtual patterns from the closure of a version space equals the virtual patterns of this version space and obviously, we obtain that $\overline{\overline{VS}} = [\bot(\overline{VS}), \top(\overline{VS})] = \overline{VS}$. Using $\bot(\overline{VS})$ and $\top(\overline{VS})$ instead of $\bot(VS)$ and $\top(VS)$ is exactly equivalent. Thus, in Section 6, we simply use virtual patterns $\bot(VS)$ and $\top(VS)$.

5.3. Virtual patterns in the primitive-based framework

This section shows that the most general and specific virtual patterns can naturally be integrated in the primitive-based framework even if they have been defined separately. But, we show that the use of $\overline{VS}$ enables us to recover the theory in VS. It is necessary because $\overline{VS}$ are used by our relaxing strategy.

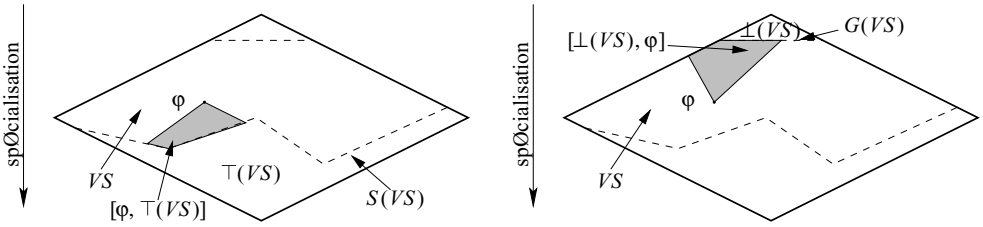


Fig. 5. Representations of intervals $[\bot(VS), \varphi]$ and $[\varphi, \top(VS)]$ on virtual lattices.

Property 6. *A monotone primitive p on VS is a monotone primitive on $\overline{VS}$.*

Proof. Let $p : \mathcal{L} \rightarrow S$ be an increasing primitive on VS. Let φ and γ be two patterns of $\overline{VS}$ such that $\varphi \prec \gamma$. First, if φ and γ belong to VS, $p(\varphi)$ is less than $p(\gamma)$ because p increases on VS. Second, if φ belongs to VS, γ is equal to $\top(VS)$. Then, as $p(\top(VS)) = \max_{\theta \in VS} p(\theta)$, we obtain that $p(\gamma) \mathfrak{e} p(\varphi)$. Third, if φ corresponds to $\bot(VS)$, φ minimalizes p: $p(\varphi) = \min_{\theta \in VS} p(\theta)$. Thereby, as γ is a simple pattern of VS or maximilizes p (by being $\top(VS)$), we have $p(\gamma) \geqslant p(\varphi)$. We conclude that p is an increasing function on $\overline{VS}$. A similar proof holds with a decreasing primitive. □

Any primitive p defined on a version space VS can be extended to its closure $\overline{VS}$. Moreover, by definition, the evaluation of this primitive p coincides on both spaces for any pattern φ in VS. Thus, the theory of q in version space VS corresponds exactly to the theory of q in its closure $\overline{VS}$ excluding the two virtual patterns $\bot(VS)$ and $\top(VS)$: $\mathcal{T}h(VS, \mathbf{r}, q) = \mathcal{T}h(\overline{VS}, \mathbf{r}, q) \backslash \{\bot(VS), \top(VS)\}$.

6. Relaxing the primitive-based constraints

6.1. Finding monotone and anti-monotone relaxations

This section depicts our relaxation method to get (anti)-monotone relaxations for the primitive-based constraints. It is achieved thanks to the joint use of the virtual patterns and the bounding operators.

The key idea of the relaxation is to benefit from the summarization due to the virtual patterns. It associates a constraint to the patterns present in the database. Intuitively, for any pattern φ, we assume that all its specializations have the most favorable specificities to satisfy the constraint. In such conditions, if no more specific pattern than φ satisfies the constraint, the anti-monotone relaxation returns $false$. The monotone relaxation is based on an analog principle. Let us consider the minimal area constraint. In the best case, a specialization of a given pattern X (i.e., $Y \supset X$) can have a frequency which equals $freq(X)$. The cardinality of Y cannot exceed the cardinality of the largest transaction of $\mathcal{D}$ (i.e., $count(\top(\mathcal{C})) = 5$). Thus, the area of any specialization is smaller than $freq(X) \times 5$ and as soon as this value is lower than the minimal area threshold, no specialization of X can satisfy the minimal area constraint.

We formalize these intuitions by means of virtual patterns and bounding operators. As the patterns have to be present in the database, they belong to the collection $\mathcal{C}$. We recall that whenever this collection contains a finite number of patterns, the latter is a version space. Thereby, we can consider its closure $\overline{\mathcal{C}}$. In this space, all the generalizations (resp. specializations) of a pattern φ are described by the interval $[\bot(\mathcal{C}), \varphi]$ (resp. $[\varphi, \top(\mathcal{C})]$), see Fig. 5. As the virtual patterns belong to the primitive-based framework

(see the previous section), the bounding operators can be applied on intervals of $\overline{\mathcal{C}}$. Then, we consider the two following primitive-based constraints for any pattern $\varphi \in \mathcal{C}$:

$$\begin{cases} \lceil q \rceil^{\perp}\langle\varphi\rangle \equiv \lceil q \rceil\langle\perp, \varphi\rangle \\ \lceil q \rceil^{\top}\langle\varphi\rangle \equiv \lceil q \rceil\langle\varphi, \top\rangle \end{cases}$$

As we mainly focus on the version space $\mathcal{C}$, the virtual patterns $\perp$ and $\top$ respectively refer to $\perp(\mathcal{C})$ and $\top(\mathcal{C})$. Theorem 1 justifies that the operators $\lceil . \rceil^{\perp}$ and $\lceil . \rceil^{\top}$ are named monotone and anti-monotone relaxing operators. It is the main result of this paper:

Theorem 1 Monotone and anti-monotone relaxations. *The primitive-based constraints $\lceil q \rceil^{\perp}$ and $\lceil q \rceil^{\top}$ are respectively a monotone and an anti-monotone relaxations of q.*

Proof. First, we prove that $\lceil q \rceil^{\top}$ is anti-monotone. Let $q \in \mathcal{Q}$ and φ be a pattern such that $\lceil q \rceil^{\top}\langle\varphi\rangle$ is $true$. Let γ be a pattern such that $\gamma \preceq \varphi$. As we have $\lceil q \rceil^{\top} = \lceil q \rceil\langle\varphi, \top\rangle$ and $[\varphi, \top] \subseteq [\gamma, \top]$, we obtain that $\lceil q \rceil\langle\varphi, \top\rangle \leqslant \lceil q \rceil\langle\gamma, \top\rangle = true$ (Lemma 1). Thus $\lceil q \rceil^{\top}$ is anti-monotone. Besides, we will check that $\lceil q \rceil^{\top}$ is a relaxation of q. Assuming that $\lceil q \rceil^{\top}\langle\varphi\rangle$ is $false$ (i.e., $\lceil q \rceil\langle\varphi, \top\rangle = false$), Property 3 gives that for any pattern $\theta \in [\varphi, \top]$ (i.e., $\varphi \preceq \theta$, we have $\lceil q \rceil\langle\varphi, \top\rangle \leqslant q(\theta) = false$. Finally, $\neg\lceil q \rceil^{\top}$ implies $\neg q$ and then, we conclude that the constraint $\lceil q \rceil^{\top}$ is an anti-monotone relaxation of q. A similar reasoning is applied on $\lceil q \rceil^{\perp}$. Thus, Theorem 1 is proved. □

Theorem 1 ensures that we achieve a monotone and an anti-monotone relaxation of any primitive-based constraint by applying the relaxing operators $\lceil . \rceil^{\perp}$ and $\lceil . \rceil^{\top}$. These relaxations inherit from the good properties of the bounding operators. As these operators deal with boolean operators, relaxations benefit from the specificities of the whole constraint thanks to the combination of the relaxations stemming from the atomical constraints. Besides, the relaxations are computed only once before the mining step as well as the bounds of the constraint.

Example 9. *Let us come back to our example of the area constraint by applying the operator $\lceil . \rceil^{\top}$ to this constraint. We have $\lceil area(X) \geqslant 6 \rceil^{\top} = freq(X) \times count(\top) \geqslant 6$ because $\lceil area(X) \geqslant 6 \rceil\langle X, Y\rangle = freq(X) \times count(Y) \geqslant 6$ (see Section 4.2) and $\lceil q \rceil^{\top}\langle X\rangle \equiv \lceil q \rceil\langle X, \top\rangle$. As $count(\top) = 4$, we obtain that $freq(X) \geqslant 6/4$ which is the anti-monotone relaxation. Symmetrically, we also deduce the monotone relaxation $count(X) < 6/5$ given in Section 2.4 stemming from $\lceil area(X) \geqslant 6 \rceil^{\perp}\langle\perp, X\rangle = freq(\perp) \times count(X) \geqslant 6 = 5 \times count(X) \geqslant 6$.*

Finally, for any primitive-based constraint, our approach automatically gives an answer to the relaxing problem stated in Section 2.4. Furthermore, constraint-based mining tasks can be performed on very difficult contexts previously unfeasible without this optimization (see Section 8).

6.2. Other version spaces

Up to now, the relaxation approach only considers virtual patterns relying on the version space of the patterns present in the database. In particular, the version space remains the same with different constraints. Sometimes, we can go further by optimizing the version space thanks to constraints in order to improve the pruning process. Assuming that we want to relax a constraint having monotone or anti-monotone atomical constraints, they can be used to restrain the version space $\mathcal{C}$. The values

corresponding to terminal primitives will be closer to the patterns and the virtual patterns stemmed from this new version space will be more accurate. Thus, the relaxations based on these virtual patterns are more selective and they again reduce the search space during the mining step. For instance, admitting that the length of the requested patterns does not exceed l, we can deal with the new version space delimited by $freq(X) \geqslant 1 \wedge count(X) \leqslant l$. Changing the collection $\mathcal{C}$ is also necessary when it is infinite in order to be able to define virtual patterns. Typically, mining an unlimited sequence of events requires to limit the length of the maximal episodes [28].

The quality of the relaxations is also shown by theoretical and practical investigations in the next sections.

7. Theoretical analysis of relaxations

This section aims at analyzing the efficiency of our relaxation approach stemmed from relaxing operators. First, we propose to detect most monotone and anti-monotone constraints. Second, we prove that these constraints are optimally relaxed.

7.1. Monotonicity testing operators

The automatic analysis of the monotone properties of a constraint is a non trivial task. In the case of the minimal frequency constraint, the anti-monotone property clearly stems from the fact that the more general a pattern is, the more frequent is. With the constraint q_6, the anti-monotone property is less intuitive and more difficult to deduce. Besides, how can it be explained that q_6 is anti-monotone and not q_1 whereas q_1 is very similar to q_6? Intuitively, the monotone properties of the primitives seem to be fundamental. Indeed, the only difference between $q_6 \equiv count(X)/freq(X) < \rho$ and $q_1 \equiv count(X) \times freq(X) < \rho$ is the arithmetical operator applied to the length and the frequency. One can note that $\times$ is an increasing function according to the second operand (on positive reals) while $/$ is a decreasing one. Similarly, there is the same difference between $+$ and $-$, and $count(X) - freq(X) < \rho$ is anti-monotone whereas $count(X) + freq(X) < \rho$ does not have good property.

Let us now formalize this intuition for any constraint q of $\mathcal{Q}$. We start by defining (anti-)monotone testing operators. Given a primitive-based constraint q, we state:

$$\lfloor q \rfloor^M = \begin{cases} true, & \text{if } \lfloor q \rfloor \langle \gamma, \varphi \rangle \text{ is equivalent to } q(\varphi) \\ false, & \text{otherwise} \end{cases}$$

$$\lceil q \rceil^M = \begin{cases} true, & \text{if } \lceil q \rceil \langle \gamma, \varphi \rangle \text{ is equivalent to } q(\varphi) \\ false, & \text{otherwise} \end{cases}$$

The operator $\lfloor . \rfloor^M$ (resp. $\lceil . \rceil^M$) is named the anti-monotone (resp. monotone) testing operator according to the specialization. Theorem 2 justifies their name:

Theorem 2 Correction of testing operators. *A primitive-based constraint satisfying* $\lfloor q \rfloor^M$ (*resp.* $\lceil q \rceil^M$) *is anti-monotone* (*resp. monotone*).

Proof. Let $q \in \mathcal{Q}$ satisfying $\lfloor q \rfloor^M$ (i.e., $\lfloor q \rfloor \langle \gamma, \varphi \rangle \equiv q(\varphi)$). Assuming that $\gamma \preceq \varphi$ and $q(\varphi) = true$, we have $\lfloor q \rfloor \langle \gamma, \varphi \rangle = q(\varphi) = true$. As $\gamma \in [\gamma, \varphi]$ and $\lfloor q \rfloor \langle \gamma, \varphi \rangle \leqslant q(\gamma)$ (Property 3), we obtain that

$q(\gamma) = true$ and conclude that q is anti-monotone. Dually, a primitive-based constraint q satisfying $\lceil q \rceil^M$ is monotone. □

Thus, Theorem 2 shows that $\lfloor . \rfloor^M$ and $\lceil . \rceil^M$ give a partial characterization based on primitives, whether anti-monotone or monotone properties. Whenever the answer of $\lfloor q \rfloor^M$ or $\lceil q \rceil^M$ is $true$, q is respectively anti-monotone and monotone. Otherwise, the answer is $false$ and nothing can be asserted about the constraint. For instance, even if q_7 is an anti-monotone constraint, we have $\lfloor q_7 \rfloor^M = false$.

Example 10 Testing monotonicity of constraints. *Applying the anti-monotone testing operator on the constraint q_6. We first compute its lower bound with the operator $\lfloor . \rfloor$. As $\geqslant$ increases in $\mathcal{B}$ according to the first variable and decreases according to the second one, we have $\lfloor freq(X)/count(X) \geqslant \rho \rfloor \langle X, Y \rangle = \lfloor freq(X)/count(X)) \rfloor \langle X, Y \rangle \geqslant \lceil \rho \rceil \langle X, Y \rangle$. As ρ is a constant, we obtain that $\lceil \rho \rceil \langle X, Y \rangle = \rho$. / increases with the first variable and decreases with the second one, we find that $\lfloor freq(X)/count(X) \rfloor \langle X, Y \rangle = \lfloor freq(X) \rfloor \langle X, Y \rangle / \lceil count(X) \rceil \langle X, Y \rangle$. Finally, $\lfloor q_6 \rfloor \langle X, Y \rangle$ is equal to $freq(Y)/count(Y) \geqslant \rho$ because $freq$ decreases and $count$ increases. We conclude that $\lfloor q \rfloor^M$ is $true$ and q_6 is anti-monotone (its lower bound on $[X, Y]$ equals to $q(Y)$). On the contrary, we obtain that $\lfloor q_1 \rfloor^M$ is equal to $false$.*

In what follows, $\lfloor \mathcal{Q} \rfloor^M$ (resp. $\lceil \mathcal{Q} \rceil^M$) designates the set of constraints which has a $true$ answer for the anti-monotone (resp. monotone) testing operator (in particular, $\{q_5, q_6\} \subset \lfloor \mathcal{Q} \rfloor^M$ and $\{q_4\} \subset \lceil \mathcal{Q} \rceil^M$). We say that $\lfloor \mathcal{Q} \rfloor^M$ and $\lceil \mathcal{Q} \rceil^M$ are respectively the set of all the detectable anti-monotone constraints and all the detectable monotone constraints. These detectable constraints verify particular properties. For instance, the bounds of a detectable anti-monotone constraint q are exact because $\lfloor q \rfloor \langle \varphi, \gamma \rangle = q(\gamma)$ and of course, γ belongs to $[\varphi, \gamma]$ (dually, $\lceil q \rceil \langle \varphi, \gamma \rangle = q(\varphi)$).

7.2. Optimal relaxations

This section theoretically proves that our relaxing approach is optimal for detectable monotone and anti-monotone constraints.

Numerous monotone (or anti-monotone) relaxations can be defined for a same constraint. But their quality depends on the size of the corresponding theory. More precisely, the closer the relaxation is to the original constraint, the better the theory of a relaxation. Thus, an optimal (anti-)monotone relaxation of a constraint is its most restrictive (anti-)monotone relaxation:

Definition 6. *A monotone* (*or anti-monotone*) *relaxation q' of a constraint q is optimal iff for any monotone* (*or anti-monotone*) *relaxation q'' of q, q'' is also a relaxation of q'.*

The optimal monotone (or anti-monotone) relaxation of q selects the smallest theory larger than that of q and having a monotone (or anti-monotone) property. A constraint has many monotone or anti-monotone relaxations, but only one optimal monotone and one anti-monotone relaxations. With the minimal frequency constraint q_5, each operator produces its optimal relaxation. Indeed, it is its own anti-monotone relaxation. On the contrary, the relaxations of minimal area constraint are not optimal. Figure 6 depicts the theory of relaxation $freq(X) \geqslant 6/4$ in gray shape. F and DE belong to the relaxation computed by our operators whereas the bold line defines a more restrictive anti-monotone relaxation.

Property 7 indicates that our approach provides the optimal (anti-)monotone relaxations for the detectable (anti-)monotone constraints:

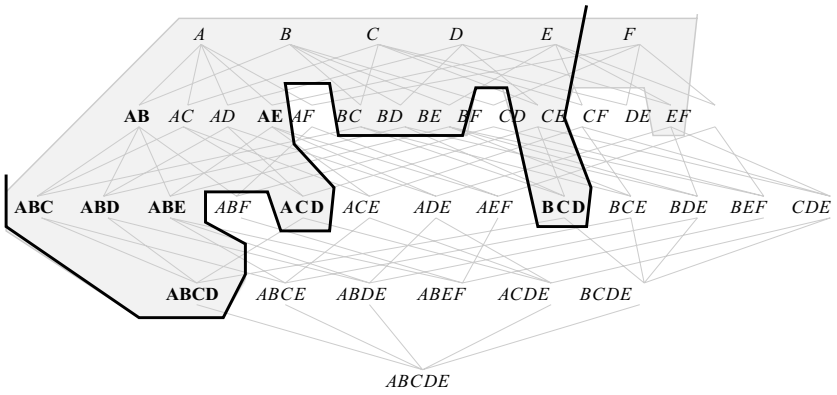


Fig. 6. Optimality of anti-monotone relaxation of minimal area constraint.

Property 7 Optimality of relaxation. *The monotone and anti-monotone relaxations* $\lceil q \rceil^{\perp}$ *and* $\lfloor q \rfloor^{\perp}$ *of a detectable monotone* (*or an anti-monotone*) *constraint* q *are optimal.*

Proof. Let $q \in \lfloor \mathcal{Q} \rfloor^{M}$. By definition of relaxing operators, we have $\lceil q \rceil^{\top}\langle\varphi\rangle \equiv \lceil q \rceil\langle\varphi, \top\rangle$ and $\lceil q \rceil^{\perp}\langle\varphi\rangle \equiv \lceil q \rceil\langle\perp, \varphi\rangle$. The definition of $\lfloor . \rfloor^{M}$ ensures that $\lceil q \rceil\langle\gamma, \varphi\rangle$ is equivalent to $q(\varphi)$. Thus, $\lceil q \rceil^{\top}\langle\varphi\rangle \equiv q(\top)$ which is a constant (often equals to $true$). This result is optimal as q is monotone, there is no better anti-monotone relaxation. Besides, $\lceil q \rceil^{\perp}\langle\varphi\rangle \equiv q(\varphi)$ i.e., the detectable mononotone constraint is its own monotone relaxation. This result is again optimal because a relaxation of q cannot be better than q. Dually, we conclude also that our relaxations of a detectable anti-monotone constraint are the optimal relaxations. □

Property 7 ensures that most of monotone and anti-monotone constraints are optimally relaxed by our operators. For other constraints, finding the optimal relaxations remains an open issue. Nevertheless, experimental analysis shows the impressive practical benefit of many non-optimal relaxations.

8. Experimental analysis

The aim of our experiments is to measure the runtime benefit brought by the anti-monotone relaxation obtained thanks to the virtual patterns. We deal with itemset and sequence databases. All the experiments were conducted on a 2.2 GHz Xeon processor with Linux operating system and 3GB of RAM memory.

8.1. Itemset mining

We use the mushroom dataset coming from the FIMI repository.[2] The constraints using numeric values were applied on a table of values randomly generated within the range [0,100]. Table 2 provides the definition of the virtual patterns $\perp$ and $\top$ with the used table of values. Experiments are based on APRIORI [2] and ECLAT [46] algorithms implemented by Borgelt's.

Figure 7 plots extractions of patterns with and without anti-monotone relaxation. On the left, the curves give runtimes for mining all the patterns satisfying the minimal area constraint according to the area

[2]http://fimi.cs.helsinki.fi/data/.

Table 2
The most general and specific virtual patterns with the mushroom dataset

Primitive p	$p(\bot)$	$p(\top)$
$freq$	8124	1
$count$	1	23
sum	0	1166
min	97	0
max	0	97

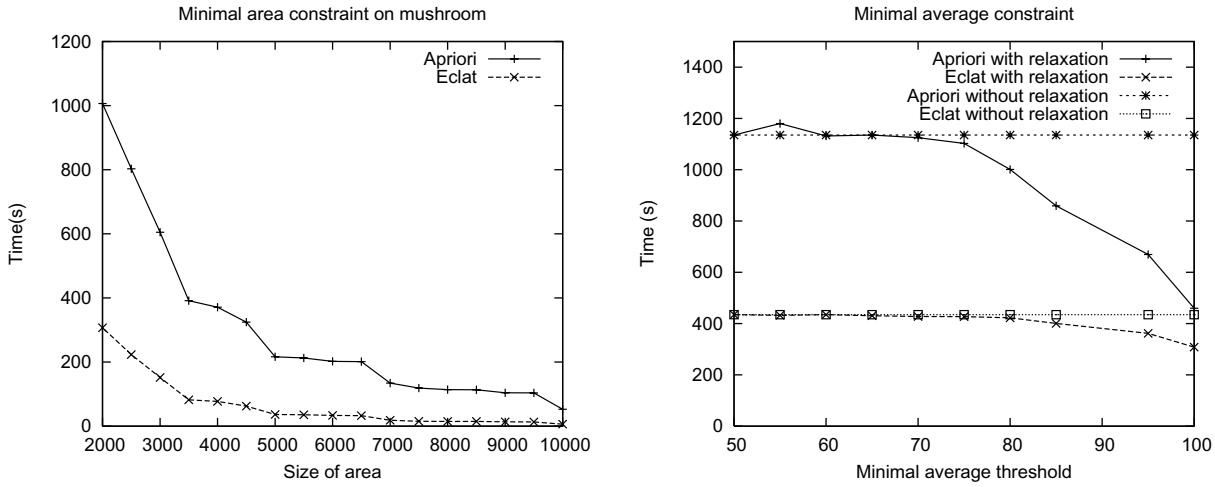


Fig. 7. Runtime performances with itemsets.

threshold. APRIORI and ECLAT curves without relaxations are not reported because their extractions fail. On the right, the curves correspond to runtime performances of mining of patterns satisfying the minimal average constraint according to the average threshold. In addition, a minimal frequency threshold of 1% is added to make feasible extractions. This threshold is used for APRIORI and ECLAT.

The curves of Fig. 7 show that in all cases, using virtual patterns drastically reduces extraction times. However, relaxations are more efficient whenever the constraint is very selective (here, when the threshold is high).

8.2. Sequential pattern mining

We now study the impact of anti-monotone relaxations on algorithms of sequential frequent pattern mining. We use two implementations: PREFIXSPAN [35] dedicated to frequent sequences and CLOSPAN [45] dedicated to closed frequent sequences. The implementations are available on the website http://illimine.cs.uiuc.edu/.

The synthetic dataset used for our experiments were generated using standard procedure described in [3]. We tested both the algorithms on dataset $C100T2.5S10I2.5$. In this dataset, the number of items is set to 1,000, and there are 10,000 sequences in the dataset. The average number of items within itemsets is set to 2.5 (denoted as $T2.5$). The average number of itemsets per sequence is set to 10 (denoted as $S10$).

Figure 8 reports runtimes needed for extracting all the sequences satisfying the minimal area constraint according to the area threshold. We recall that the area definition is $freq(X) \times count(X)$ where $freq(X)$

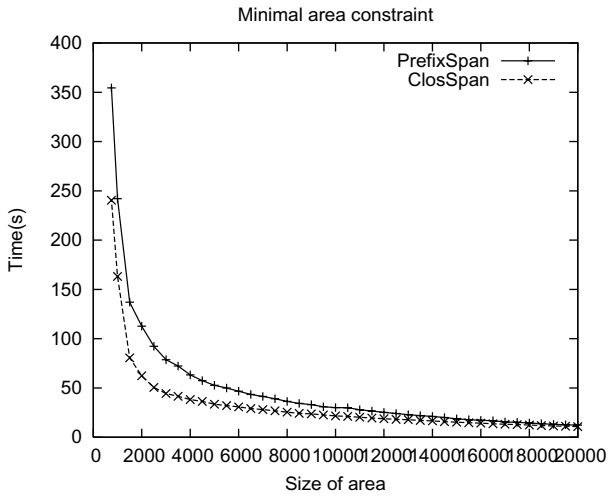


Fig. 8. Runtime performances with sequences.

is the frequency of X and $count(X)$ is the number of items belonging to X (e.g., $count(\langle(AB)(A)\rangle) = 3$). As for itemset mining, without a minimal frequency constraint, the mining is unfeasible and then, the approach without relaxation fails.

Efficiency of anti-monotone relaxations for sequence mining is comparable to performances observed with itemsets. The efficiency of the pruning depends on the selectivity of the constraint. The more selective the constraint is, the faster the mining is.

The above experiments demonstrate that the large scope of our relaxing approach. First, the relaxations stemmed from our method are useful for different kinds of languages such as itemsets (in Section 8.1) or sequences (in Section 8.2). Second, the relaxed constraints are independent from the choice of the mining algorithm. The extraction step can be efficiently performed with a breadth-first search (e.g., APRIORI) or a depth-first search manner (e.g., ECLAT and PREFIXSPAN). Finally, note that prototypes as CLOSPAN extract only closed patterns whereas any kind of pattern according the primitive-based framework are successfully addressed by our method.

9. Conclusion

In this paper, we have presented a very general approach for constraint-based pattern mining relying on any partially ordered language describing the patterns. The user is able to define in a flexible way a large set of useful constraints. The primitive-based constraints are a super-class of classes such as monotone, anti-monotone, convertible and succinct constraints. We introduced the notion of virtual patterns to summarize the specificities of the data mining context in order to automatically achieve monotone and anti-monotone relaxations. Efficient pruning conditions stemmed from these relaxations and usual constraint mining algorithms can be reused to push them whereas the constraint does not satisfy monotonicity properties. Experiments show that these relaxations are very efficient for reducing the search space during the mining step leading to mining tasks which were unfeasible before. As expected, the more selective the constraint is, the faster the mining. More generally, our automated method has several assets. Firstly, we demonstrate that computed relaxations are optimal with detectable monotone and anti-monotone constraints. Secondly, this pre-processing approach on the constraint is

independent from the choice of the algorithm. In practice, many existing algorithms can use these relaxations to mine different kinds of patterns under constraints.

Further work addresses the refinement of the most specific and general virtual patterns to take into account more accurately the contiguous subsets and supersets. In the same way, we would like to revisit mining problems by introducing other virtual patterns which appear to be an elegant and efficient trick.

Acknowledgments

This work has been partially supported by the ANR (French Research National Agency) project Bingo2 ANR-07-MDCO-014 which is a follow-up of the first Bingo project (2004–2007).

References

[1] R. Agrawal, T. Imielinski and A. N. Swami. Mining association rules between sets of items in large databases, in: *SIGMOD Conference*, P. Buneman and S. Jajodia, eds, ACM Press, 1993, pp. 207–216.

[2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases, in: *VLDB*, J.B. Bocca, M. Jarke and C. Zaniolo, eds, Morgan Kaufmann, 1994, pp. 487–499.

[3] R. Agrawal and R. Srikant, Mining sequential patterns, in: *ICDE*, P.S. Yu and A.L.P. Chen, eds, IEEE Computer Society, 1995, pp. 3–14.

[4] C. Antunes and A.L. Oliveira. Constraint relaxations for discovering unknown sequential patterns, in: *KDID 2004, Knowledge Discovery in Inductive Databases, Proceedings of the Third International Workshop on Knowledge Discovery inInductive Databases, Pisa, Italy, September 20, 2004, Revised Selected and Invited Papers, volume 3377 of Lecture Notes in Computer Science*, B. Goethals and A. Siebes, eds, Springer, 2004, pp. 11–32.

[5] R.J. Bayardo, The hows, whys, and whens of constraints in itemset and rule discovery, in: *Proc. of the Workshop on Inductive Databases and Constraint Based Mining* (*IDW'05*), 2005.

[6] S. Bistarelli and F. Bonchi, Interestingness is not a dichotomy: Introducing softness in constrained pattern mining, in: *PKDD, volume 3721 of Lecture Notes in Computer Science*, A. Jorge, L. Torgo, P. Brazdil, R. Camacho and J. Gama, eds, Springer, 2005, pp. 22–33.

[7] F. Bonchi, F. Giannotti, A. Mazzanti and D. Pedreschi, Exante: Anticipated data reduction in constrained pattern mining, in: *PKDD, volume 2838 of Lecture Notes in Computer Science*, N. Lavrac, D. Gamberger, H. Blockeel and L. Todorovski, eds, Springer, 2003, pp. 59–70.

[8] F. Bonchi and C. Lucchese, On closed constrained frequent pattern mining, in: *Proceedings of the 4th IEEE International Conference on Data Mining* (*ICDM 2004*), Brighton, UK, IEEE Computer Society, 1–4 November 2004, pp. 35–42.

[9] F. Bonchi and C. Lucchese, Pushing tougher constraints in frequent pattern mining, in: *Advances in Knowledge Discovery and Data Mining, 9th Pacific-Asia Conference, PAKDD 2005, Hanoi, Vietnam, May 18–20, 2005, Proceedings, volume 3518 of Lecture Notes in Computer Science*, T.B. Ho, D. Cheung and H. Liu, eds, Springer, 2005, pp. 114–124.

[10] C. Bucila, J. Gehrke, D. Kifer and W. White, DualMiner: A dual-pruning algorithm for itemsets with constraints, in: *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Edmonton, Alberta, Canada, ACM, July 23–26, 2002.

[11] G. Dong and J. Li, Efficient mining of emerging patterns: discovering trends and differences, in: *Proceedings of the fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (*KDD'99*), New York, NY, USA, ACM Press, 1999, pp. 43–52.

[12] M. El-Hajj and O.R. Zaïane, Non-recursive generation of frequent k-itemsets from frequent pattern tree representations, in: *Data Warehousing and Knowledge Discovery, 5th International Conference, DaWaK 2003, Prague, Czech Republic, September 3–5, 2003, Proceedings, volume 2737 of Lecture Notes in Computer Science*, Y. Kambayashi, M.K. Mohania and W. Wöß, eds, Springer, 2003, pp. 371–380.

[13] M. El-Hajj, O.R. Zaïane and P. Nalos, Bifold constraint-based mining by simultaneous monotone and anti-monotone checking, in: *Proceedings of the 5th IEEE International Conference on Data Mining* (*ICDM 2005*), *27–30 November 2005*, Houston, Texas, USA, IEEE Computer Society, 2005, pp. 146–153.

[14] J. Fischer, *Version Spaces in Constraint-based Data Mining*, Master thesis, University of Freiburg, 2003.

[15] K. Gade, J. Wang and G. Karypis, Efficient closed pattern mining in the presence of tough block constraint, in: *ACM SIGKDD International Conference on Knowledge Discovery in Databases*, 2004.

[16] M.N. Garofalakis, R. Rastogi and K. Shim, SPIRIT: Sequential pattern mining with regular expression constraints, *The VLDB Journal* (1999), 223–234.

[17] F. Geerts, B. Goethals and T. Mielikäinen, Tiling databases, in: *Proceedings of the 7th International Conference on Discovery Science* (*DS'04*), *volume 3245*, LNAI, 2004, pp. 278–289.
[18] A. Giacometti, D. Laurent and C.T. Diop, Condensed representations for sets of mining queries, in: *Proceedings of KDID'02*, 2002.
[19] D. Gunopulos, R. Khardon, H. Mannila and H. Toivonen, Data mining, hypergraph transversals, and machine learning, in: *PODS*, ACM Press, 1997, pp. 209–216.
[20] J. Han, J. Pei and Y. Yin, Mining frequent patterns without candidate generation, in: *SIGMOD Conference*, W. Chen, J.F. Naughton and P.A. Bernstein, eds, ACM, 2000, pp. 1–12.
[21] T. Imielinski and H. Mannila, A database perspective on knowledge discovery, in: *Communication of the ACM*, 1996, pp. 58–64.
[22] B. Jeudy and F. Rioult, Database transposition for constrained closed pattern mining, in: *Proceedings of Third International Workshop on Knowledge Discovery in Inductive Databases* (*KDID*) *co-located with ECML/PKDD*, 2004.
[23] J. Klema, S. Blachon, A. Soulet, B. Crémilleux and O. Gandrillon, Constraintbased knowledge discovery from sage data, *In Silico Biology* **8** (2008), 157–175.
[24] D.E. Knuth, *The Art of Computer Programming*, Third volumes, AddisonWesley, 1968.
[25] S. Kramer, L.D. Raedt and C. Helma, Molecular feature mining in hiv data, in: *KDD*, 2001, pp. 136–143.
[26] S.D. Lee and L.D. Raedt, An algebra for inductive query evaluation, in: *Proceedings of the Second International Workshop on Inductive Databases* (*KDID'03*), Rudjer Boskovic Institute, Zagreb, Croatia, September 2003, pp. 80–96.
[27] H. Mannila and H. Toivonen, Levelwise search and borders of theories in knowledge discovery, *Data Min Knowl Discov* **1**(3) (1997), 241–258.
[28] H. Mannila, H. Toivonen and A.I. Verkamo, Discovering frequent episodes in sequences, in: *KDD*, 1995, pp. 210–215.
[29] T.M. Mitchell, Generalization as search, *Artif Intell* **18**(2) (1982), 203–226.
[30] S. Morishita and J. Sese, Traversing itemset lattice with statistical metric pruning, in: *PODS*, ACM, 2000, pp. 226–236.
[31] R.T. Ng, L.V.S. Lakshmanan, J. Han and A. Pang, Exploratory mining and pruning optimizations of constrained association rules, in: *SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data, June 2–4, 1998*, L.M. Haas and A. Tiwary, eds, Seattle, Washington, USA, ACM Press, 1998, pp. 13–24.
[32] N. Pasquier, Y. Bastide, R. Taouil and L. Lakhal, Discovering frequent closed itemsets for association rules, in: *Proceedings of 7th International Conference on Database Theory* (*ICDT'99*), *volume 1540 of Lecture notes in artifial intelligence*, Jerusalem, Israel, Springer Verlag, 1999, pp. 398–416.
[33] J. Pei, J. Han and L.V.S. Lakshmanan, Mining frequent item sets with convertible constraints, in: *ICDE*, 2001, pp. 433–442.
[34] J. Pei, J. Han and R. Mao, CLOSET: An efficient algorithm for mining frequent closed itemsets, in: *ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, 2000, pp. 21–30.
[35] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal and M. Hsu, Prefixspan: Mining sequential patterns by prefix-projected growth, in: *ICDE*, IEEE Computer Society, 2001, pp. 215–224.
[36] J. Pei, J. Han and W. Wang, Mining sequential patterns with constraints in large databases, in: *CIKM*, ACM, 2002, pp. 18–25.
[37] L.D. Raedt and S. Kramer, The levelwise version space algorithm and its application to molecular fragment finding, in: *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI 2001, Seattle, Washington, USA, August 4-10, 2001*, B. Nebel, ed., Morgan Kaufmann, 2001, pp. 853–862.
[38] A. Soulet and B. Crémilleux, An efficient framework for mining flexible constraints, in: *Advances in Knowledge Discovery and Data Mining, 9th Pacifific-Asia Conference, PAKDD 2005, Hanoi, Vietnam, May 18–20, 2005, Proceedings, volume 3518 of Lecture Notes in Computer Science*, T.B. Ho, D. Cheung and H. Liu, eds, Springer, 2005, pp. 661–671.
[39] A. Soulet and B. Crémilleux, Exploiting virtual patterns for automatically pruning the search space, in: *Knowledge Discovery in Inductive Databases, 4th International Workshop, KDID 2005, Porto, Portugal, October 3, 2005, Revised Selected and Invited Papers, volume 3933 of Lecture Notes in Computer Science*, F. Bonchi and J.-F. Boulicaut, eds, Springer, 2005, pp. 202–221.
[40] A. Soulet and B. Crémilleux, Optimizing constraint-based mining by automatically relaxing constraints, in: *Proceedings of the 5th IEEE International Conference on Data Mining* (*ICDM 2005*), *27-30 November 2005*, Houston, Texas, USA, IEEE Computer Society, 2005, pp. 777–780.
[41] R. Srikant, Q. Vu and R. Agrawal, Mining association rules with item constraints, in: *KDD*, 1997, pp. 67–73.
[42] A. Termier, M.-C. Rousset and M. Sebag, Dryade: A new approach for discovering closed frequent trees in heterogeneous tree databases, in: *Proceedings of the 4th IEEE International Conference on Data Mining* (*ICDM 2004*), *1–4 November 2004*, Brighton, UK, IEEE Computer Society, 2004, pp. 543–546.
[43] J. Wang and J. Han, BIDE: Efficient mining of frequent closed sequence, in: *ICDE*, IEEE Computer Society, 2004, pp. 79–90.
[44] K. Wang, Y. Jiang, J.X. Yu, G. Dong and J. Han, Divide-and-approximate: A novel constraint push strategy for iceberg cube mining, *IEEE Trans Knowl Data Eng* **17**(3) (2005), 354–368.

[45] X. Yan, J. Han and R. Afshar, CloSpan: Mining closed sequential patterns in large databases, in: *Proceedings of the Third SIAM International Conference on Data Mining*, D. Barbará and C. Kamath, eds, San Francisco, CA, USA, SIAM, May 1–3, 2003.
[46] M.J. Zaki, Scalable algorithms for association mining, *IEEE Trans Knowl Data Eng* **12**(2) (2000), 372–390.
[47] M.J. Zaki, N. Parimi, N. De, F. Gao, B. Phoophakdee, J. Urban, V. Chaoji, M.A. Hasan and S. Salem, Towards generic pattern mining, in: *Formal Concept Analysis, Third International Conference, ICFCA 2005, Lens, France, February 14–18, 2005, Proceedings, volume 3403 of Lecture Notes in Computer Science*, B. Ganter and R. Godin, eds, Springer, 2005, pp. 1–20.