An Efficient Framework for Mining Flexible Constraints

Arnaud Soulet and Bruno Crémilleux

GREYC, CNRS - UMR 6072, Université de Caen,
Campus Côte de Nacre, F-14032 Caen Cédex France
`Forename.Surname@info.unicaen.fr`

Abstract. Constraint-based mining is an active field of research which is a key point to get interactive and successful KDD processes. Nevertheless, usual solvers are limited to particular kinds of constraints because they rely on properties to prune the search space which are incompatible together. In this paper, we provide a general framework dedicated to a large set of constraints described by `SQL`-like and syntactic primitives. This set of constraints covers the usual classes and introduces new tough and flexible constraints. We define a pruning operator which prunes the search space by automatically taking into account the characteristics of the constraint at hand. Finally, we propose an algorithm which efficiently makes use of this framework. Experimental results highlight that usual and new complex constraints can be mined in large datasets.

1 Introduction

Mining patterns under various kinds of constraints is a key point to get interactive and successful KDD processes. There is a large collection of constraints which are useful for the user and the latter needs an independent tool to tackle various and flexible queries. The outstandingly useful constraint of frequency is often used in practice. Furthermore, we have efficient algorithms to extract patterns satisfying it. Nevertheless, in the context of constraint-based mining, supplementary constraints like interestingness measures or syntactic constraints have to be added to achieve relevant and desired patterns. The number of elementary constraints and their combinations is too important to build a particular solver dedicated to each constraint. These observations are sound motivations to design and implement a general solver which is able to mine, in a flexible way, patterns checking various and meaningful constraints.

Let us recall that constraint-based mining remains a challenge due to the huge size of the search space which has to be explored (it exponentially increases according to the number of features of the data). Classical algorithms are based on pruning properties in order to reduce the search space. But, unfortunately, these properties are deeply linked to the constraints and many constraints (e.g., average [13], variance [10], growth rate [6]) have been studied individually [1]. Section 2.2 overviews the main classes of constraints and recalls that efficient algorithms are devoted to some of these classes. Several approaches [16, 2] are based on condensed representations of frequent patterns which are easier to mine. We will see that our work uses this approach but without limitation to the classical classes of constraints. The paradigm of inductive databases [8] proposes to handle constraints by reusing existing algorithms (for instance, a complex constraint is decomposed into several constraints having suitable properties like monotonicity). However, the decomposition and the optimization of constraints remain non-trivial tasks [5]. To the best of our knowledge, there is no existing framework presenting at the same time flexibility and effective computation.

In this paper, we present a new general framework to soundly and completely mine patterns under a constraint specified by the user as a simple parameter. We think that this framework brings three meaningful contributions. First, it allows a large set of constraints: the constraints are described by combinations of SQL-like aggregate primitives and syntactic primitives (see Section 3.1). This formalism deals with the most usual constraints (e.g., monotonous, anti-monotonous and convertible ones) and allows to define more original new constraints (e.g., the area constraint which is on the core of our running example). Furthermore, this formalism also enables to combine constraints with boolean operators. Second, thanks to an automatic process to compute lower and upper bounds of a constraint on an interval and a general *pruning operator*, the constraint is pushed in the extraction step. Finally, we provide an algorithm called Music (Mining with a User-SpecifIed Constraint) which allows the practical use of this framework. Music guarantees an efficient pruning to offer short run-time answers and facilitate the iterative process of KDD. We developed a prototype to implement this algorithm.

This paper is organized in the following way. Section 2 introduces the basic notations and related work. A running example (i.e., the area constraint) shows the common difficulties of constraint-based mining and the key ideas of our framework. Section 3 depicts the set of constraints that we address, details the theoretical framework and defines the pruning operator. Section 4 indicates how to use it by providing the Music algorithm. Finally, Section 5 presents experimental results showing the efficiency of Music on various constraints.

2 Context and Motivations

2.1 Notation

Let us first introduce the basic notations. A transactional dataset $\mathcal{D}$ is a triplet $(\mathcal{A}, \mathcal{O}, R)$ where $\mathcal{A}$ is a set of attributes, $\mathcal{O}$ is a set of objects and $R \subseteq \mathcal{A} \times \mathcal{O}$ is a binary relation between the attributes and the objects. $(a, o) \in R$ expresses that the object o has the attribute a (see for instance Table 1 where $A, \ldots, F$ denote the attributes and $o_1, \ldots, o_6$ denote the objects). A pattern X is a subset of attributes.

The aim of constrained patterns mining is to extract all patterns present in $\mathcal{D}$ and checking a predicate q. The minimal frequency constraint is likely the most usual one (the frequency of a pattern X is the number of objects in $\mathcal{D}$ that contain X, i.e. $count(X) \geq \gamma$ where γ is a threshold). Many algorithms since [1] efficiently mine this constraint by using closure or free (or key) patterns [3, 14].

Table 1. Example of a transactional data set

$\mathcal{D}$

Objects	Attributes
o_1	A B E F
o_2	A E
o_3	A B C D
o_4	A B C D E
o_5	D E
o_6	C F

2.2 Related Work

Many works have been done with various complex constraints (e.g., average [13], variance [10], growth rate [6]) providing particular approaches. More generally, we can distinguish several classes of constraints. A well-known class of constraints is based on *monotonicity*. A constraint q is anti-monotone (resp. monotone) according to the specialization of the attributes if whenever $X \subseteq Y$ then $q(Y) \Rightarrow q(X)$ (resp. $q(X) \Rightarrow q(Y)$). For instance, the minimal frequency constraint is anti-monotonous. In this case, the search space can be efficiently pruned by a general level-wise algorithm [12]. Another class is the *convertible* constraints. Such a constraint uses an ordering relation on the attributes in order to obtain properties of monotonicity on the prefixes [15] (typically, the minimal average constraint q_8 is convertible, see Section 3.1 for its exact definition). Let us note that Wang et al. introduce in [18] a method dedicated to the aggregate constraints (e.g. the minimal frequency constraint or the average q_8).

There are specific algorithms devoted to these different classes of constraints. Unfortunately, the combination of constraints may require again a particular algorithm. For example, a conjunction of two convertible constraints may lead to a no convertible constraint, and a particular algorithm has to been developed. So, several approaches attempt to overcome these difficulties. The *inductive databases* framework [8] proposes to decompose complex constraints into several constraints having good properties like monotonicity. This approach needs to apply non-trivial reformulations and optimizations of constraints [5]. Introduced in [10], the concept of *witness* provides properties to simultaneously prune patterns under different kinds of constraints. Nevertheless, this approach does not propose a method to automatically obtain witnesses. Thus, instead of building a particular algorithm to mine patterns, a particular algorithm to find witnesses is needed. By exploiting equivalence classes (i.e., a set of patterns having the same outcome with respect to the constraint), condensed representations [4] enable powerful pruning criteria during the extraction which greatly improve the efficiency of algorithms [3, 14]. But only few works exploit the equivalence classes with monotonous and anti-monotonous classes [2] or other constraints [9, 16]. Our work follows this approach but it addresses a much more general set of constraints (see Section 3.1).

2.3 Problem Statement and Key Idea of Our Framework

Let us come back on the example given by Table 1. Assume that we are interested in all subsets of $\mathcal{A}$ having an *area* greater than 4 i.e. $count(X) \times length(X) \geq 4$ (where *length* is the cardinality of the pattern). Recently, [7] have dedicated efficient approaches to only mine the closed patterns that check this constraint. The constraint of area is difficult because it is neither monotone ($area(A) \geq 4$ but $area(ABF) < 4$), nor anti-monotone ($area(B) < 4$ but $area(AB) \geq 4$), nor convertible (no ordering relation exists). None decomposition of this constraint benefits from the properties of these classes of constraints. Thus, the practical approach is to mine all patterns with their own frequency and then to post-process them by checking the constraint on each pattern. Unfortunately, this method fails with large datasets due to a too much number of candidate patterns.

From this running example, we now indicate how to take into account the characteristics of the constraint to present our pruning strategy of the search space. The main idea is based on the definition of lower and upper bounds of the constraint on an interval, the latter after allows the pruning. We can notice that if $X \subseteq Z \subseteq Y$, the area of the pattern Z can be bounded by $count(Y) \times length(X) \leq count(Z) \times length(Z) \leq count(X) \times length(Y)$. We note that if $count(Y) \times length(X) \geq 4$, the area of the pattern Z is larger than 4 and Z checks the constraint. In this example, with $X = AB$ and $Y = ABCD$, the area of $count(ABCD) \times length(AB)$ is equal to 4 and the patterns AB, ABC, ABD, $ABCD$ have an area larger than 4. Thus, it is not necessary to check the constraint for these four patterns. Similarly, when $count(X) \times length(Y)$ is strictly smaller than 4, the area of the pattern Z ($X \subseteq Z \subseteq Y$) is inevitably smaller than 4. In these two cases, the interval $[X, Y]$ can be pruned for this constraint. Also, the patterns AB, ABC, ABD and $ABCD$, which are included between AB and $ABCD$, satisfy the constraint. Instead of outputting these four patterns, it is more judicious to only output the corresponding interval $[AB, ABCD]$. This one can be seen as a condensed representation of the patterns with respect to the constraint. This idea - mining a *representation* of the constrained patterns - is generalized in the next section to a large set of constraints.

3 Pruning the Search Space by Pruning an Interval

3.1 The Set of Constraints

Our work deals with the set of constraints $\mathcal{Q}$ recursively defined by Table 2. Examples of constraints of $\mathcal{Q}$ are given at the end of this section. We claim that $\mathcal{Q}$ defines a very large set of constraints.

$\mathcal{L}_\mathcal{A}$ (resp. $\mathcal{L}_\mathcal{O}$) denotes the language associated with the attributes $\mathcal{A}$ (resp. the objects $\mathcal{O}$) i.e. the powerset $2^\mathcal{A}$ (resp. the powerset $2^\mathcal{O}$). The set of constraints $\mathcal{Q}$ is based on three spaces: the booleans $\mathfrak{B}$ (i.e., *true* or $false$), the positive reals $\Re^+$ and the patterns of $\mathcal{L} = \mathcal{L}_\mathcal{A} \cup \mathcal{L}_\mathcal{O}$. In addition to the classical operators of these domains, the function *count* denotes the frequency of a pattern, and *length* its cardinality. Given a function $val : \mathcal{A} \cup \mathcal{O} \rightarrow \Re^+$, we extend it to a pattern X and note $X.val$ the set $\{val(a)|a \in X\}$. This kind of function is used with

Table 2. Set of constraints $\mathcal{Q}$

Constraint $q \in \mathcal{Q}$	Operator(s)	Operand(s)
$q_1 \theta q_2$	$\theta \in \{\wedge, \vee\}$	$(q_1, q_2) \in \mathcal{Q}^2$
θq_1	$\theta \in \{\neg\}$	$q_1 \in \mathcal{Q}$
$e_1 \theta e_2$	$\theta \in \{<, \leq, =, \neq, \geq, >\}$	$(e_1, e_2) \in \mathcal{E}^2$
$s_1 \theta s_2$	$\theta \in \{\subset, \subseteq, =, \neq, \supseteq, \supset\}$	$(s_1, s_2) \in \mathcal{S}^2$
constant $b \in \mathfrak{B}$	-	-

Aggregate expression $e \in \mathcal{E}$	Operator(s)	Operand(s)
$e_1 \theta e_2$	$\theta \in \{+, -, \times, /\}$	$(e_1, e_2) \in \mathcal{E}^2$
$\theta(s)$	$\theta \in \{count, length\}$	$s \in \mathcal{S}$
$\theta(s.val)$	$\theta \in \{sum, max, min\}$	$s \in \mathcal{S}$
constant $r \in \Re^+$	-	-

Syntactic expression $s \in \mathcal{S}$	Operator(s)	Operand(s)
$s_1 \theta s_2$	$\theta \in \{\cup, \cap, \setminus\}$	$(s_1, s_2) \in \mathcal{S}^2$
$\theta(s_1)$	$\theta \in \{f, g\}$	$s_1 \in \mathcal{S}$
variable $X \in \mathcal{L}_\mathcal{A}$	-	-
constant $l \in \mathcal{L} = \mathcal{L}_\mathcal{A} \cup \mathcal{L}_\mathcal{O}$	-	-

the usual SQL-like primitives *sum*, *min* and *max*. For instance, $sum(X.val)$ is the sum of *val* of each attribute of X. Finally, f is the intensive function i.e. $f(O) = \{a \in \mathcal{A} | \forall o \in O, (a, o) \in R\}$, and g is the extensive function i.e. $g(A) = \{o \in \mathcal{O} | \forall a \in A, (a, o) \in R\}$. We give now some examples of constraints belonging to $\mathcal{Q}$ and highlighting the generality of our framework.

$q_1 \equiv count(X) \geq \gamma \times |\mathcal{D}|$
$q_2 \equiv count(X) \times length(X) \geq 2500$
$q_3 \equiv X \subseteq A$
$q_4 \equiv X \supseteq A$
$q_5 \equiv length(X) \geq 10$
$q_6 \equiv sum(X.val) \geq 500$
$q_7 \equiv max(X.val) < 50$
$q_8 \equiv sum(X.val)/length(X) \geq 50$
$q_9 \equiv min(X.val) \geq 30 \wedge max(X.val) \leq 90$
$q_{10} \equiv max(X.val) - min(X.val) \leq 2 \times length(X)$
$q_{11} \equiv length(X \setminus A) > length(X \cap A)$
$q_{12} \equiv q_2 \wedge q_3$
$q_{13} \equiv q_6 \vee \neg q_5$
$q_{14} \equiv q_5 \wedge q_7$

Starting from a constraint of $\mathcal{Q}$, the following sections explain how to get sufficient conditions to prune the search space.

3.2 Bounding a Constraint on an Interval

This section indicates how to automatically compute lower and upper bounds of a constraint of $\mathcal{Q}$ on an interval without enumerating each pattern included in the interval. These bounds will be used by the pruning operator (see Section 3.3).

Let X and Y be two patterns. The interval between these patterns (denoted $[X, Y]$) corresponds to the set $\{Z \in \mathcal{L}_\mathcal{A} | X \subseteq Z \subseteq Y\}$. In our running example dealing with the area constraint, Section 2.3 has shown that $count(Y) \times length(X)$ and $count(X) \times length(Y)$ are respectively a lower bound and an upper bound of the constraint for the patterns included in the interval $[X, Y]$. At a higher level, one can also notice that $\forall Z \in [X, Y]$, we have $(count(Y) \times length(X) \geq 4) \leq q(Z) \leq (count(X) \times length(Y) \geq 4)$ with respect

Table 3. The definitions of $\lfloor.\rfloor$ and $\lceil.\rceil$

$e \in \mathcal{E}_i$	Operator(s)	$\lfloor e \rfloor\langle X, Y\rangle$	$\lceil e \rceil\langle X, Y\rangle$
$e_1\theta e_2$	$\theta \in \{\wedge, \vee, +, \times, \cup, \cap\}$	$\lfloor e_1 \rfloor\langle X, Y\rangle\theta\lfloor e_2 \rfloor\langle X, Y\rangle$	$\lceil e_1 \rceil\langle X, Y\rangle\theta\lceil e_2 \rceil\langle X, Y\rangle$
$e_1\theta e_2$	$\theta \in \{>, \geq, \supset, \supseteq, -, /, \backslash\}$	$\lfloor e_1 \rfloor\langle X, Y\rangle\theta\lceil e_2 \rceil\langle X, Y\rangle$	$\lceil e_1 \rceil\langle X, Y\rangle\theta\lfloor e_2 \rfloor\langle X, Y\rangle$
θe_1	$\theta \in \{\neg, count, f, g\}$	$\theta\lceil e_1 \rceil\langle X, Y\rangle$	$\theta\lfloor e_1 \rfloor\langle X, Y\rangle$
$\theta(e_1.val)$	$\theta \in \{min\}$	$\theta(\lceil e_1 \rceil\langle X, Y\rangle.val)$	$\theta(\lfloor e_1 \rfloor\langle X, Y\rangle.val)$
$\theta(e_1)$	$\theta \in \{length\}$	$\theta\lfloor e_1 \rfloor\langle X, Y\rangle$	$\theta\lceil e_1 \rceil\langle X, Y\rangle$
$\theta(e_1.val)$	$\theta \in \{sum, max\}$	$\theta(\lfloor e_1 \rfloor\langle X, Y\rangle.val)$	$\theta(\lceil e_1 \rceil\langle X, Y\rangle.val)$
$c \in E_i$	-	c	c
$X \in \mathcal{L}_\mathcal{A}$	-	X	Y

to $false < true$. Thus, the area constraint is bounded on the interval. Those bounds only depend on the patterns X and Y and their definitions are the same for any interval $[X, Y]$.

Let us generalize this approach for any constraint q of $\mathcal{Q}$. For that, we define two operators denoted $\lfloor.\rfloor$ and $\lceil.\rceil$ (see Table 3). Starting from q and $[X, Y]$, the recursive application of these operators leads to compute one boolean with $\lfloor.\rfloor$ (noted $\lfloor q \rfloor\langle X, Y\rangle$) and one boolean with $\lceil.\rceil$ (noted $\lceil q \rceil\langle X, Y\rangle$). Property 1 will show that $\lfloor q \rfloor\langle X, Y\rangle$ (resp. $\lceil q \rceil\langle X, Y\rangle$) is a lower bound (resp. an upper bound) of the interval $[X, Y]$ for q. In other words, these operators enable to automatically compute lower and upper bounds of $[X, Y]$ for q. This result stems from the properties of increasing and decreasing functions. In Table 3, the general notation E_i designates one space among $\mathfrak{B}$, $\Re^+$ or $\mathcal{L}$ and $\mathcal{E}_i$ the associated expressions (for instance, the set of constraints $\mathcal{Q}$ for the booleans $\mathfrak{B}$). Several operators given in Table 2 must be split into several operators of Table 3. For instance, the equality $e_1 = e_2$ is decomposed to $(e_1 \leq e_2) \wedge (e_1 \geq e_2)$. In Table 3, the functions are grouped by monotonous properties according to their variables. For instance, the operators $-$, $/$ and $\backslash$ are increasing functions according to the first variable and decreasing functions according to the second variable.

Let us illustrate $\lfloor.\rfloor$ and $\lceil.\rceil$ on the area constraint: $\lfloor count(X) \times length(X) \geq 4 \rfloor\langle X, Y\rangle = \lfloor count(X) \times length(X) \rfloor\langle X, Y\rangle \geq \lceil 4 \rceil\langle X, Y\rangle = \lfloor count(X) \rfloor\langle X, Y\rangle \times \lfloor length(X) \rfloor\langle X, Y\rangle \geq 4 = count(\lceil X \rceil\langle X, Y\rangle) \times length(\lfloor X \rfloor\langle X, Y\rangle) \geq 4 = count(Y) \times length(X) \geq 4$. Symmetrically, $\lceil count(X) \times length(X) \geq 4 \rceil\langle X, Y\rangle$ is equal to $count(X) \times length(Y) \geq 4$.

Property 1 shows that $\lfloor q \rfloor$ and $\lceil q \rceil$ are bounds of the constraint q.

Property 1 (bounds of an interval). *Let q be a constraint, $\lfloor q \rfloor$ and $\lceil q \rceil$ are respectively a lower bound and an upper bound of q i.e. given an interval $[X, Y]$ and a pattern Z included in it, we have $\lfloor q \rfloor\langle X, Y\rangle \leq q(Z) \leq \lceil q \rceil\langle X, Y\rangle$.*

This property justifies that $\lfloor.\rfloor$ and $\lceil.\rceil$ are respectively named the lower and upper bounding operators (due to space limitation the proof is not given here, see [17]). Contrary to most frameworks, these top-level operators allow us to directly use constraints containing conjunctions or disjunctions of other constraints. Besides, they compute quite accurate bounds. In the particular case of monotonous constraints, these bounds are even *exact.* They have other meaningful properties (linearity or duality) which are not developed here (details in [17]).

3.3 Pruning Operator

In this section, we define an operator, starting from a constraint q, which provides a condition to safely prune or not an interval. As for the area constraint, there are two different strategies to prune an interval $[X, Y]$ by using the bounds of q. If a lower bound of q on $[X, Y]$ is equal to *true* (i.e., the lower bound checks q), all the patterns included in $[X, Y]$ check q because they are all greater than *true*. We say that we *positively* prune the patterns of $[X, Y]$. Conversely, we can *negatively* prune $[X, Y]$ when an upper bound is $false$ because all the patterns of $[X, Y]$ do not check q. Note that witnesses [10] already exploit these two kinds of pruning. We define now the pruning condition for q and the pruning operator.

Definition 1 (pruning operator). *Let q be a constraint, the pruning condition for q, denoted by $[q]$, is equal to $\lfloor q \rfloor \vee \neg \lceil q \rceil$. $[\centerdot]$ is called the pruning operator.*

This definition is linked to the two ways of pruning: $\lfloor q \rfloor$ is associated with the positive pruning, and $\neg \lceil q \rceil$ to the negative one. For instance, the pruning condition for the area constraint on an interval $[X, Y]$ is $(count(Y) \times length(X) \geq 4) \wedge (count(X) \times length(Y) < 4)$. This conjunction corresponds to the two cases allowing us to prune intervals (see Section 2.3).

The following key theorem will be used extensively.

Theorem 1. *Let q be a constraint and $[X, Y]$ an interval, if $[q]\langle X, Y\rangle$ is true, then all the patterns included in $[X, Y]$ have the same value for q.*

Due to space limitation, the proof is not proposed here (see [17]).

Whenever the pruning condition is *true* on an interval $[X, Y]$, we know the value of any pattern of $[X, Y]$ by checking only one pattern. Thereby, $[X, Y]$ can be pruned without having to check the constraint on whole patterns of $[X, Y]$. Section 4 details how to prune the search space with the pruning condition.

Let us note that the converse of Theorem 1 is false because $\lfloor \centerdot \rfloor$ (resp. $\lceil \centerdot \rceil$) does not give the greatest lower (resp. the least upper) bound. However, in practice, the pruning operator often provides powerful pruning (see Section 5).

4 Music: A Constraint-Based Mining Algorithm

Music (Mining with a User-SpecifIed Constraint) is a level-wise algorithm which takes advantage of the pruning operator to efficiently mine constrained patterns and get a representation of these patterns. It takes one constraint q belonging to $\mathcal{Q}$ as input and one additional anti-monotonous constraint q_{AM} to benefit from the usual pruning of level-wise algorithm [1] (line 4). The completeness with respect to q is ensured by sticking *true* for q_{AM}. Music returns in output all the intervals containing the patterns checking $q \wedge q_{AM}$. Music is based on tree key steps: the creating of the generators similar to one used in [1] (line 14), the evaluation of candidates by scanning the dataset in order to compute the extension of patterns (line 3), and finally the testing candidates (lines 7-12). Its main originality is that the "tested candidates" are different from generators and they are intervals instead of patterns.

```
MUSIC (a constraint $q \in \mathcal{Q}$, an anti-monotonous constraint $q_{AM}$, a dataset $\mathcal{D}$)

 1. $\mathcal{C}and_1 := \{a \in \mathcal{A} | q_{AM}(a) = true\}$ ; $\mathcal{R}es := \emptyset$ ; $\mathcal{AC}and_1 := \emptyset$ ; $k := 1$
 2. while $\mathcal{C}and_k \cup \mathcal{AC}and_k \neq \emptyset$ do
 3.     for each $X \in \mathcal{C}and_k$ do $X.extension := \{o \in \mathcal{O} | X \subseteq f(o)\}$
 4.     $\mathcal{G}en_k := \{X \in \mathcal{C}and_k | q_{AM}(X) = true \text{ and } X \text{ is } free\}$
 5.     $\mathcal{AC}and_{k+1} := \emptyset$
 6.     for each $X \in \mathcal{G}en_k \cup \mathcal{AC}and_k$ do
 7.         if $[q]\langle X, f(X.extension)\rangle = true$ then
 8.             if $q(X) = true$ then $\mathcal{R}es := \mathcal{R}es \cup \{[X, f(X.extension)]\}$
 9.         else do
10.             if $q(X) = true$ then $\mathcal{R}es := \mathcal{R}es \cup \{[X, X]\}$
11.             $\mathcal{AC}and_{k+1} := \mathcal{AC}and_{k+1} \cup \{X \cup \{a\} | a \in h(X) \backslash X\}$
12.         od
13.     od
14.     $\mathcal{C}and_{k+1} :=$ APRIORI-GEN($\mathcal{G}en_k$)
15.     $k := k + 1$
16. od
17. return $\mathcal{R}es$
```

MUSIC uses intervals already proposed in [11], where the left bound is a generator (i.e., a free pattern or an additionnal candidate) and the right one, its closure (i.e., a closed pattern). The closed patterns are exactly the fixed points of the closure operator $h = f \circ g$. An important property on the extension stems from the closure: $g(X) = g(h(X))$. Moreover, as the closure operator is extensive, any pattern is a subset of its closure and the interval $[X, h(X)]$ has always a sense. The pruning condition is pushed into the core of the mining by applying it on the intervals defined above. Such an approach enables a powerful pruning criterion during the extraction thanks to the use of an anti-monotonous constraint based on the *freeness* [3] (line 4). If an interval $[X, h(X)]$ satisfies the pruning condition, all the patterns in $[X, h(X)]$ are definitively pruned. Otherwise, some patterns of $[X, h(X)]$ are added as *additional* candidates to repeat the same process on shorter intervals (line 11). The computation of the extension for each pattern (line 3) is sufficient to deduce the values of all the primitives given by Table 2 even if they depend on the dataset like g or *count*. In particular, the frequency of a pattern X is $length(g(X))$ and the closure of X is computed with $f(g(X))$. Note that the additionnal candidates have a low cost because they belong to $[X, h(X)]$ and their extension is equal to $g(h(X))$.

Due to the lack of place, we provide here only an intuitive proof of the correction of MUSIC (see [17] for a formal proof). All the patterns are comprise between a free pattern and its closure. As MUSIC covers all the free patterns, all the intervals $[X, h(X)]$ are checked by the pruning condition. There are two cases. First, if the pruning condition is true, all the patterns included in $[X, h(X)]$ are checked. Otherwise, the patterns included in $[X, h(X)]$ are enumerated level by level until that the interval between it and its closure can be pruned (that always arises because $[q]\langle X, X\rangle = true$). Thus, the whole search space is covered.

5 Experimental Results

The aim of our experiments is to measure the run-time benefit brought by our framework on various constraints (i.e., constraints $q_1, \ldots, q_{14}$ defined in Section 3.1). All the tests were performed on a 700 GHz Pentium III processor with Linux and 1Go of RAM memory. The used dataset is the version of mushroom coming from the FIMI repository[1]. The constraints using numeric values were applied on attribute values (noted *val*) randomly generated within the range [1,100]. We compare our algorithm with an APRIORI-like approach (i.e., mining all patterns according to q_{AM} and using a filter to select patterns checking q).

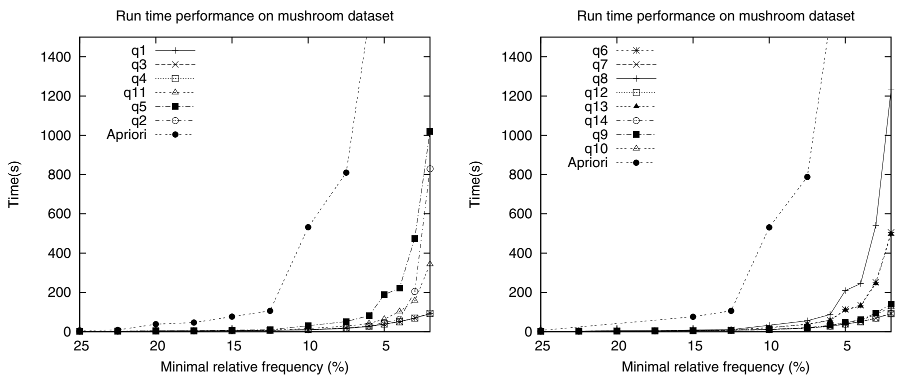

Fig. 1. Mining patterns with various constraints on mushroom dataset

Figure 1 plots the comparison between MUSIC and APRIORI run-times according to $q_1, \ldots, q_{14}$. With the constraint q_1, MUSIC has no additional candidates and its behavior is similar than [3]. That shows the abilities of MUSIC towards usual constraints. The best performances are achieved by the constraints q_3, q_4, q_7, q_9 and q_{10} because the number of additional candidates in these cases is very low. On the other hand, the three worst performances (i.e., q_2, q_5 and q_8) are obtained with the constraints including *length*. It is interesting to observe that the run-time performances are independent of the complexity of the constraint (i.e., the number of combinations). For instance, a very complex constraint such as q_{10} is quickly mined and a conjunction of constraint such as $q_2 \wedge q_3$ has better results than q_2 alone. This fact can be explained with the improvement of the selectivity of the constraint. Additional results are given in [17]. The good

[1] `http://fimi.cs.helsinki.fi/data/`

behavior of Music with complex constraints allows the user to ask a broad set of queries and to mine constrained patterns which were intractable until now.

6 Conclusion

In this paper, we have proposed a new and general framework to efficiently mine patterns under constraints based on SQL-like and syntactic primitives. This framework deals with boolean combinations of the usual constraints and allows to define new complex constraints. The efficiency of the approach relies on the pruning of the search space on intervals which are took into account by a general pruning operator. Starting from this approach, Music algorithm mines soundly and completely patterns under a primitive-based constraint given by the user as a simple parameter. The experimental results show that Music clearly outperforms Apriori with all constraints. New tough constraints can be mined in large datasets. We think that our algebraisation is an important step towards the integration of the constraint-based mining in database systems.

Further work addresses optimization of specific primitives like *length*. About the generality of our framework, we would like also to know if other primitives used to define constraints should be useful to achieve successful KDD processes from real world data set. We think that our ongoing work on geographical datasets is a good way to test new expressive queries specified by a geographer expert and the usefulness of the primitives.

Acknowledgements. This work has been partially founded by the ACI "Bingo".

References

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, pages 432–444, 1994.

[2] F. Bonchi and C. Lucchese. On closed constrained frequent pattern mining. In *proceedings of ICDM'04*, pages 35–42, 2004.

[3] J. F. Boulicaut, A. Bykowski, and C. Rigotti. Free-sets: a condensed representation of boolean data for the approximation of frequency queries. *Data Mining and Knowledge Discovery journal*, 7(1):5–22, 2003.

[4] T. Calders and B. Goethals. Minimal k-free representations of frequent sets. In *proceedings of PKDD'03*, pages 71–82. Springer, 2003.

[5] L. De Raedt, M. Jäger, S. D. Lee, and H. Mannila. A theory of inductive query answering. In *proceedings of ICDM'02*, pages 123–130, Maebashi, Japan, 2002.

[6] G. Dong and J. Li. Efficient mining of emerging patterns: Discovering trends and differences. In *Knowledge Discovery and Data Mining*, pages 43–52, 1999.

[7] K. Gade, J. Wang, and G. Karypis. Efficient closed pattern mining in the presence of tough block constraints. In *proceedings of ACM SIGKDD*, pages 138–147, 2004.

[8] T. Imielinski and H. Mannila. A database perspective on knowledge discovery. In *Communication of the ACM*, pages 58–64, 1996.

[9] B. Jeudy and F. Rioult. Database transposition for constrained (closed) pattern mining. In *proceedings of KDID'04*, pages 89–107, 2004.

[10] D. Kiefer, J. Gehrke, C. Bucila, and W. White. How to quickly find a witness. In *proceedings of ACM SIGMOD/PODS 2003 Conference*, pages 272–283, 2003.

[11] M. Kryszkiewicz. Inferring knowledge from frequent patterns. In *proceedings of Soft-Ware 2002*, Lecture Notes in Computer Science, pages 247–262, 2002.

[12] H. Mannila and H. Toivonen. Levelwise search and borders of theories in knowledge discovery. *Data Mining and Knowledge Discovery*, 1(3):241–258, 1997.

[13] R. T. Ng, L. V. S. Lakshmanan, and J. Han. Exploratory mining and pruning optimizations of constrained associations rules. In *proceedings of SIGMOD*, 1998.

[14] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Discovering frequent closed itemsets for association rules. *Lecture Notes in Computer Science*, 1999.

[15] J. Pei, J. Han, and L. V. S. Lakshmanan. Mining frequent item sets with convertible constraints. In *proceedings of ICDE*, pages 433–442, 2001.

[16] A. Soulet, B. Crémilleux, and F. Rioult. Condensed representation of emerging patterns. In *proceedings of PAKDD'04*, pages 127–132, 2004.

[17] A. Soulet and B. Crémilleux. A general framework designed for constraint-based mining. Technical report, Université de Caen, Caen, France, 2004.

[18] K. Wang, Y. Jiang, J. X. Yu, G. Dong, and J. Han. Pushing aggregate constraints by divide-and-approximate. In *proceedings of ICDE*, pages 291–302, 2003.