Optimizing Constraint-Based Mining by Automatically Relaxing Constraints

Arnaud Soulet Bruno Crémilleux
GREYC, CNRS - UMR 6072, Université de Caen
Campus Côte de Nacre
F-14032 Caen Cédex France
{Forename.Surname}@info.unicaen.fr

Abstract

In constraint-based mining, the monotone and anti-monotone properties are exploited to reduce the search space. Even if a constraint has not such suitable properties, existing algorithms can be re-used thanks to an approximation, called relaxation. In this paper, we automatically compute monotone relaxations of primitive-based constraints. First, we show that the latter are a superclass of combinations of both kinds of monotone constraints. Second, we add two operators to detect the properties of monotonicity of such constraints. Finally, we define relaxing operators to obtain monotone relaxations of them.

1 Introduction

Mining patterns under constraints is a significant field of research in data mining. Many constraints like interestingness measures or syntactic constraints are useful to achieve relevant patterns. Although the constraint is chosen by the user, the mining step has to be automated in order to facilitate the KDD process. In particular, to handle the constraint should be independent from the user.

Many algorithms reduce the search space of patterns by using the monotonicity property [9]. Unfortunately, most of the constraints are neither monotone, nor anti-monotone. Nevertheless, such "difficult" constraints can be extracted by re-using available algorithms. Indeed, the constraint is approximated by another having suitable monotone property [5, 4]. The collection of extracted patterns corresponding to this approximation has to be a superset of the collection satisfying the original constraint. Thereby, in a post-processed step, a basic filtering selects the desired patterns. The approximated constraint is called *relaxation*. To the best of our knowledge, there is no existing theoretical work which proposes a general approach to obtain relaxations having monotone properties. A specialist of data mining or a mathematician can compute a relaxation of a given constraint. However, a process or a common user have not the ability to find it.

In this paper, we propose three main contributions. At first, we show that the *primitive-based constraints* are a superclass of both kinds of monotone constraints and their combinations. Secondly, we propose an original approach to detect monotone constraints by applying new formal operators, named the *monotone and anti-monotone testing operators*. Finally, we provide a monotone and an anti-monotone relaxations for each primitive-based constraint, these relaxations are automatically obtained by the *monotone and anti-monotone relaxing operators*.

This paper is organized in the following way. Section 2 introduces the basic notations and related works. Section 3 depicts the set of constraints that we address and links them to the monotone constraints. Section 4 defines the operators which allows us to detect monotonicity. Finally, these operators are exploited to relax constraints in Section 4.

2 Context and related works

Basic definitions 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. A pattern is a set of attributes. The aim of constrained patterns mining is to extract all patterns present in $\mathcal{D}$ and satisfying a predicate q (also called query or constraint). Many predicates have a particular property named monotonicity. A constraint q is monotone (resp. anti-monotone) according to the specialization iff whenever $X \subseteq Y$ then $q(X) \Rightarrow q(Y)$ (resp. $q(Y) \Rightarrow q(X)$). The set of monotone (resp. anti-monotone) constraints is denoted by $\mathcal{Q}_M$ (resp. $\mathcal{Q}_{AM}$). Let us remark that we often use "(anti-)monotone" instead of "monotone and anti-monotone" when that is not ambiguous.

Table 1 provides several examples of constraints with their properties about monotonicity (i.e., $\mathcal{Q}_M$ and $\mathcal{Q}_{AM}$). In general, a constraint is not monotone, neither anti-

Constraint	$\mathcal{Q}_M$	$\mathcal{Q}_{AM}$
$q_1(X) \equiv count(X) \geq \gamma$		$\times$
$q_2(X) \equiv length(X)/count(X) < \rho$		$\times$
$q_3(X) \equiv A \subseteq X$	$\times$	
$q_4(X) \equiv sum(X.val) \geq \gamma$	$\times$	
$q_5(X) \equiv length(X) \times count(X) < \rho$		
$q_6(X) \equiv (q_1(X) \vee q_2(X)) \wedge q_3(X)$		
$q_7(X) \equiv 2 \times length(X) - length(X) \leq \rho$		$\times$

monotone (e.g., the constraints q_5 or q_6). Let us note that the function $count$ denotes the frequency of a pattern (i.e., the number of objects in $\mathcal{D}$ that contain X), and $length$ its cardinality. Given a function $val : \mathcal{A} \rightarrow \Re^+$, we extend it to a pattern X and note $X.val$ the multiset $\{val(a)|a \in X\}$. This kind of function is used with the usual SQL-like primitives sum, min and max. For instance, $sum(X.val)$ is the sum of val of each attribute of X.

In this paper, we focus on relaxing constraints:

Definition 1 (relaxed constraint) *Let q be a constraint, a constraint q' is a relaxed constraint of q iff q' is always satisfied whenever q is satisfied i.e., $q \Rightarrow q'$.*

Starting from a user-specified constraint q, we want to automatically define a monotone constraint q_M and an anti-monotone one q_{AM} which are relaxations of q, i.e. $q \Rightarrow q_M$ and $q \Rightarrow q_{AM}$. For instance, the constraints $q_3 \in \mathcal{Q}_M$ and $(q_1 \vee q_2) \in \mathcal{Q}_{AM}$ are relaxed constraints of q_6.

Related work In [9], the authors defines the notion of (anti-)monotone constraints, but they do not provide a characterization of this notion. In particular, the monotonicity of a constraint cannot be deduced from primitives. Given an (anti-)monotone constraint, the search space can be efficiently pruned by a general level-wise algorithm [1, 9]. Many usual and useful constraints are anti-monotone (e.g., *freeness* or q_1) or monotone constraints according to the specialization (e.g., q_3 and q_4). There are specific algorithms devoted to mine a combination of one monotone constraint and one anti-monotone constraint according to the specialization [3, 2]. More generally, other particular constraints [7] or classes of constraints [10, 11, 12], use an additional monotone constraint in order to focus on interesting patterns and to improve the mining. The *inductive databases* framework [6] proposes to decompose complex constraints into several constraints having suitable properties like monotonicity. Based on version spaces, an algebra is proposed to evaluate and optimize such inductive queries [8]. Nevertheless, to the best of our knowledge, there is no existing theoretical work which proposes a general approach to identify monotone properties or to relax inductive queries.

3 Scope of the primitive-based constraints

We briefly recall the framework of the primitive-based constraints that we have introduced in [12] by giving a more general definition.

Contrary to the usual classes of constraints, the definition of primitive-based constraints is based on a set of primitives defined as below:

Definition 2 (primitive) *A function $p : S_{i_1} \times \cdots \times S_{i_n} \rightarrow S_j$ is a primitive of the primitive-based framework iff for each variable, p is monotone function (when the others remain constant).*

The set of primitives is denoted by $\mathcal{P}$. Let us note that Definition 2 implies that the domains $S_{i_1} \times \cdots \times S_{i_n}$ and S_j are partially or totally ordered sets. In this paper, the used primitives (see Section 2) are based on three spaces: the booleans $\mathfrak{B}$ (i.e. $true$ or $false$), the positive reals $\Re^+$ and the patterns of $\mathcal{L}_\mathcal{A}$, where $\mathcal{L}_\mathcal{A}$ denotes the language associated with the attributes $\mathcal{A}$ i.e. the power-set $2^\mathcal{A}$. These different spaces are ordered sets: $false < true$ for booleans, usual ordering relation for reals and the inclusion operator for sets.

In practice, more complex primitives (not monotone) are useful to the user (e.g., the average). They can be seen as a high-level primitive. The next definition provides the set of all possible high-level primitives starting from $\mathcal{P}$:

Definition 3 (high-level primitive) *The high-level primitives of degree n, denoted by $\mathcal{H}_n$, is recursively defined by:*

- *if $n = 0$: $\mathcal{H}_0$ is the set of the primitives $\mathcal{P}$ defined on $\mathcal{L}_\mathcal{A}$.*
- *if $n > 0$: $\mathcal{H}_n$ is the set of functions h such that $h = p(h_1, \ldots, h_k)$ where $p \in \mathcal{P}$ of arity k and $\forall i \in \{1, \ldots, k\}$, $h_i \in \mathcal{H}_{n_i}$, with $max_{i \in \{1,\ldots,k\}} n_i = n - 1$. $p(h_1, \ldots, h_k)$ is named the decomposition of h.*

In the following, the set of whole high-level primitives is noted $\mathcal{H}$ i.e. $\mathcal{H} = \bigcup_{i=0}^{\infty} \mathcal{H}_i$.

A primitive-based constraint is a constraint which is a high-level primitive of $\mathcal{H}$:

Definition 4 (primitive-based constraint) *A constraint $q : \mathcal{L}_\mathcal{A} \rightarrow \mathfrak{B}$ is a primitive-based constraint iff q is a high-level primitive of $\mathcal{H}$.*

A primitive-based constraint $q : \mathcal{L}_\mathcal{A} \rightarrow \mathfrak{B}$ is a combination of monotone primitives. This set of constraints is denoted by $\mathcal{Q}$. Then, we have $\mathcal{Q} = \{q : \mathcal{L}_\mathcal{A} \rightarrow \mathfrak{B} | q \in \mathcal{H}\}$. All the constraints given by Table 1 belong to $\mathcal{Q}$.

This section shows that the primitive-based constraints compose a superclass of both kinds of monotone constraints and their boolean combinations. It means that it is a general framework.

In the above section, the primitive-based constraints coming from our chosen primitives (e.g., $count$ or $length$) allow us to define numerous and varied (anti-)monotone constraints (e.g., constraints $q_1, \ldots, q_4$ belong to $\mathcal{Q}$). Nevertheless, there are (anti-)monotone constraints which cannot be decomposed into such primitives. For instance, the freeness cannot be defined directly.

In practice, the following property proves that our solver can always be extended to a particular (anti-)monotone constraint:

Property 1 *Let q be an (anti-)monotone constraint according to the specialization, q is a monotone primitive.*

In other words, Property 1 expresses that all the monotone constraints according to specialization or generalization are primitives of our framework. Our class of primitive-based constraints $\mathcal{Q}$ is more general than the class of (anti-)monotone constraints i.e., $\mathcal{Q}_M \cup \mathcal{Q}_{AM} \subseteq \mathcal{Q}$. Furthermore, as $\{\wedge, \vee, \neg\} \subset \mathcal{P}$, the next property even proves that $\mathcal{Q}$ is larger than boolean combinations of (anti-)monotone constraints.

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

This property ensures that a conjunction or disjunction of primitive-based constraints is also a primitive-based one. In particular, inductive queries like constraint q_6, are a subset of the primitive-based constraints. This shows that our framework is adapted to treat them.

Thus, in Sections 4 and 5, we can handle inductive queries and more complex constraints to identify some interesting monotone properties and to deduce relaxations.

4 Detecting monotone properties

This section defines new operators on the primitive-based framework in order to detect some potential monotone properties of a primitive-based constraint.

The automatic analysis of the monotone properties of a constraint is not a trivial task. In the case of the minimal frequency constraint, the anti-monotone property clearly stems from the fact that the shorter a pattern is, the more frequent it is. With the constraint q_2, the anti-monotone property is less intuitive and more difficult to deduce. Besides, how to explain that q_2 is anti-monotone and not q_5 whereas q_5 is very similar to q_2? Intuitively, the monotone properties of the primitives seem to be fundamental. Indeed, the only difference between $q_2 \equiv length(X)/count(X) < \rho$ and $q_5 \equiv length(X) \times count(X) < \rho$ 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 the positive reals) while / is a decreasing one. Let us now formalize this intuition for any constraint q of $\mathcal{Q}$. We start by giving the definition of (anti-)monotone testing operators.

Definition 5 (testing operators) *Let h be a high-level primitive, the booleans $\lfloor h \rfloor^M$ and $\lceil h \rceil^M$ are defined by:*

- *if $\deg h = 0$: $\lfloor h \rfloor^M = true$ iff h decreases and $\lceil h \rceil^M = true$ iff h is increases.*
- *if $\deg h > 0$: $\lfloor h \rfloor^M = b_1 \wedge \cdots \wedge b_k$ and $\lceil h \rceil^M = B_1 \wedge \cdots \wedge B_k$ where $p(h_1, \ldots, h_k)$ is the decomposition of h and for each variable $i \in \{1, \ldots, k\}$:*

$$\begin{cases} b_i = \lfloor h_{n_i} \rfloor^M \text{ and } B_i = \lceil h_{n_i} \rceil^M & \text{if } p \text{ increases with the } i^{th} \text{ variable} \\ b_i = \lceil h_{n_i} \rceil^M \text{ and } B_i = \lfloor h_{n_i} \rfloor^M & \text{otherwise} \end{cases}$$

According to Theorem 1, $\lfloor . \rfloor^M$ (resp. $\lceil . \rceil^M$) is named the anti-monotone (resp. monotone) testing operator according to the specialization.

Theorem 1 (correction of testing operators) *Let q be a primitive-based constraint, if $\lfloor q \rfloor^M$ (resp. $\lceil q \rceil^M$) is equal to $true$, then q is an anti-monotone (resp. a monotone) constraint according to the specialization.*

Thus, Theorem 1 gives a partial characterization of both kinds of monotone properties based on primitives. Given a constraint, whenever the answer of $\lfloor q \rfloor^M$ or $\lceil q \rceil^M$ is $true$, we know the monotone property of this constraint. 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 $\lfloor q_7 \rfloor^M = false$.

Let us come back on the constraint q_2 given by Table 1 in order to verify the anti-monotone property thanks to the operator $\lfloor . \rfloor^M$: $\lfloor length(X)/count(X) < \rho \rfloor^M = \lceil length(X)/count(X) \rceil^M \wedge \lfloor \rho \rfloor^M = \lceil length(X) \rceil^M \wedge \lfloor count(X) \rfloor^M \wedge true = true \wedge true = true$. Now, we can test the anti-monotone property on the constraint q_5: $\lfloor q_5(X) \rfloor^M = \lfloor length(X) \times count(X) < \rho \rfloor^M = \lceil length(X) \times count(X) \rceil^M \wedge \lfloor \rho \rfloor^M = \lceil length(X) \rceil^M \wedge \lceil count(X) \rceil^M \wedge true = true \wedge false = false$. In the same way, we obtain that $\lceil q_5 \rceil^M$ is equal to $false$.

In the rest of paper, $\lfloor \mathcal{Q} \rfloor^M$ (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,

$\{q_1, q_2\} \subset \lfloor \mathcal{Q} \rfloor^M$ and $\{q_3, q_4\} \subset \lceil \mathcal{Q} \rceil^M$). Thereby, Theorem 1 and the negative examples q_7 and $\neg q_9$ ensure that $\lfloor \mathcal{Q} \rfloor^M \subset \mathcal{Q}_{AM}$ and $\lceil \mathcal{Q} \rceil^M \subset \mathcal{Q}_M$.

The next section provides an adequate using of (anti-)monotone testing operators to relax constraints.

5 Relaxing primitive-based constraints

This section provides two new operators which automatically relax a constraint to achieve a relaxed constraint satisfying monotone properties.

By using the (anti-)monotone testing operators developed in the previous section, we want to take into account the properties of monotonicity in order to compute (anti-)monotone relaxations. This approach relies on the behavior of relaxation stemming from the boolean combinations. We start by giving the definition of (anti-)monotone relaxing operators:

Definition 6 ((anti-)monotone relaxing operators) *Let q be a primitive-based constraint, the constraints $\lfloor q \rfloor^R$ and $\lceil q \rceil^R$ are recursively defined by ($\theta \in \{\wedge, \vee\}$):*

$$\lfloor q \rfloor^R = \begin{cases} q, & \text{if } \lfloor q \rfloor^M = true \\ \lfloor q_1 \rfloor^R \theta \lfloor q_2 \rfloor^R, & \text{if } \lfloor q \rfloor^M = false \text{ and } q = q_1 \theta q_2 \\ true, & \text{otherwise} \end{cases}$$

$$\lceil q \rceil^R = \begin{cases} q, & \text{if } \lceil q \rceil^M = true \\ \lceil q_1 \rceil^R \theta \lceil q_2 \rceil^R, & \text{if } \lceil q \rceil^M = false \text{ and } q = q_1 \theta q_2 \\ true, & \text{otherwise} \end{cases}$$

For instance, the anti-monotone relaxation of q_6 is $\lfloor (q_1 \vee q_2) \wedge q_3 \rfloor^R = \lfloor (q_1 \vee q_2) \rfloor^R \wedge \lfloor q_3 \rfloor^R = (q_1 \vee q_2) \wedge true = q_1 \vee q_2$. Analogously, $\lceil q_6 \rceil^R$ is equal to q_3. Let us note that the constraints $\lfloor q_6 \rfloor^R$ and $\lceil q_6 \rceil^R$ correspond to the monotone relaxations proposed in Section 2.

Now, we give the most important result about relaxing operators:

Theorem 2 (correction of relaxing operators) *Let $q \in \mathcal{Q}$, the constraint $\lfloor q \rfloor^R$ (resp. $\lceil q \rceil^R$) is an anti-monotone (resp. a monotone) relaxed constraint of q.*

This theorem justifies that $\lfloor . \rfloor^R$ and $\lceil . \rceil^R$ are respectively named the anti-monotone relaxing operator and the monotone relaxing operator. Let us note that the operator $\lfloor . \rfloor^R$ (resp. $\lceil . \rceil^R$) is defined from $\mathcal{Q}$ to $\lfloor \mathcal{Q} \rfloor^M$ (resp. $\lceil \mathcal{Q} \rceil^M$).

We propose a practical method stemming from (anti-)monotone relaxing operators. In our primitive-based framework [12], we propose a constraint solver named MUSIC. The latter mines soundly and completely patterns under a primitive-based constraint q. Nevertheless, it can benefit from an anti-monotone constraint q_{AM} to improve the extraction. In such a context, we implement a constraint relaxer in order to take into account the user-specified constraint q. It identifies the adequate anti-monotone constraint q_{AM}. Thus, starting from a simple parameter, it has a twofold advantage: improving the efficiency of the mining process without complicating it for the user.

6 Conclusion

In this paper, we have proposed a general approach for automatically relaxing constraints belonging to the primitive-based framework. We show that this set of primitive-based constraints is a superclass of both kinds of monotone constraints and deals with boolean combinations. We defined two new operators which allow to detect monotone or anti-monotone constraints. These operators identify relevant parts of the constraint with regard to monotone properties. These parts are then combined and finally we get a monotone and an anti-monotone relaxed constraints. They can be efficiently pushed in the extraction stage. This improves the KDD process without requiring a particular theoretical knowledge of the user.

Acknowledgements. This work has been partially funded by the ACI "masse de données" (MD 46, CNRS 2004-2007) BINGO (Bases de données INductives pour la Gén0mique).

References

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. of VLDB*, pages 487–499, 1994.
[2] F. Bonchi and C. Lucchese. On closed constrained frequent pattern mining. In *Proc. of ICDM*, pages 35–42, 2004.
[3] C. Bucila, J. Gehrke, D. Kifer, and W. White. Dualminer: A dual-pruning algorithm for itemsets with constraints. In *Proc. of SIGKDD*, pages 42–51, 2002.
[4] A. L. O. Claudia Antunes. Mining patterns using relaxations of user defined constraints. In *Post-proc. of KDID*, 2004.
[5] M. N. Garofalakis, R. Rastogi, and K. Shim. SPIRIT: Sequential pattern mining with regular expression constraints. In *Proc. of VLDB*, pages 223–234, 1999.
[6] T. Imielinski and H. Mannila. A database perspective on knowledge discovery. *Comm. Of The Acm*, 1996.
[7] D. Kiefer, J. Gehrke, C. Bucila, and W. White. How to quickly find a witness. In *Proc. of SIGMOD/PODS*, 2003.
[8] S. D. Lee and L. D. Raedt. An algebra for inductive query evaluation. In *Proc. of KDID*, pages 80–96, 2003.
[9] H. Mannila and H. Toivonen. Levelwise search and borders of theories in knowledge discovery. *Data Mining and Knowledge Discovery*, 1(3):241–258, 1997.
[10] R. Ng, L. V. S. Lakshmanan, J. Han, and T. Mah. Exploratory mining via constrained frequent set queries, 1999.
[11] J. Pei, J. Han, and L. V. S. Lakshmanan. Mining frequent item sets with convertible constraints. In *Proc. of ICDE*, pages 433–442, 2001.
[12] A. Soulet and B. Crémilleux. An efficient framework for mining flexible constraints. In *Proc. of PAKDD*, pages 661–670, 2005.