Actes des 16ème Journées Bases de Données Avancées BDA'00, Blois (F),
24-27 Octobre 2000. pp. 221-237.

Using Constraints During Set Mining: Should We Prune or not?

Jean-François Boulicaut and Baptiste Jeudy

Institut National des Sciences Appliquées de Lyon
Laboratoire d'Ingénierie des Systèmes d'Information
Bâtiment 501
F-69621 Villeurbanne cedex, France
Tél. +33(0)472438905 - Fax. +33(0)472438713
{Jean-Francois.Boulicaut,Baptiste.Jeudy}@lisi.insa-lyon.fr

Abstract. Knowledge discovery in databases (KDD) is an interactive process that can be considered from a querying perspective. Within the inductive database framework, an inductive query on a database is a query that might return generalizations about the data e.g., frequent itemsets, association rules, data dependencies. To study evaluation schemes of such queries, we focus on the simple case of (frequent) itemset mining and consider the effective use of constraints (selection criteria for desired generalizations) during the mining phase. Roughly speaking, levelwise algorithms that have been proved effective for frequent itemset mining iterate on (a) candidate support count, (b) candidate generation, and (c) candidate safe pruning. Using a generic algorithm, we discuss possibilities for "pushing" constraints into the discovery algorithm. If it is rather easy for the so-called anti-monotone constraints (e.g., the frequency constraint), it becomes harder for constraints that are not anti-monotone. We show an important tradeoff: pushing these latter constraints can avoid expensive anti-monotone constraint checking but might lead to less effective pruning phases. As a consequence, a "generate and test" strategy is sometimes better. Among others, we discuss the interesting case of monotone constraints (negation of anti-monotone constraints). Formalizing the use of constraints suggests new directions of research for a generic approach to inductive query evaluation.

Keywords: datamining, knowledge discovery in databases, algorithm, Apriori, Close.

1 Introduction

Current technology makes it fairly easy to collect data, but data analysis tends to be slow and expensive. Consequently, finding tools for data mining, i.e., semiautomatic methods for locating interesting information in the masses of unanalyzed or underanalyzed data, has become an important research area [6].

Considering a data mining process as a sequence of queries over the data but also generalizations of the data, the so-called *theory* of the data, has been more or less explicitly used for various mining tasks [8, 11]. Given a language $\mathcal{L}$ of patterns (e.g., association rules, data dependencies), the theory of a database $\mathbf{r}$ with respect to $\mathcal{L}$ and a selection predicate q is the set $Th(\mathbf{r}, \mathcal{L}, q) = \{\phi \in \mathcal{L} \mid q(\mathbf{r}, \phi)\}$. The predicate q indicates whether a pattern ϕ is considered interesting (e.g., ϕ denotes a property that is "frequent" in $\mathbf{r}$, or a "surprising" property w.r.t. some unexpectedness objective measure). The selection predicate can be defined as a conjunction of atomic constraints $\mathcal{C}$ that have to be satisfied by the patterns. Some of its conjuncts refer to the "behavior" of a pattern in the data (e.g., its

"frequency" in a given dataset is above a user-given threshold), some others define syntactical restrictions on desired patterns (e.g., its "length" is below a user-given threshold). During a KDD process, many related theories have to be computed. Therefore, we need query languages that enable the user to select subsets of data as well as tightly specified theories. This gave rise to the concept of *inductive databases*, i.e., databases that contain intensionally defined theories in addition to the usual data. This framework has been suggested in the seminal paper [8], a rough formalization was proposed in [10] and has been refined in [4, 5].

In this paper, we consider inductive queries that return sets $S \subseteq \texttt{Items}$ where `Items` denotes a collection of attributes[1].

Example 1. A simple mining task: assume one wants to find all frequents itemsets that contain attribute `A` (where $\texttt{Items} = \{\texttt{A}, \texttt{B}, \texttt{C}, \texttt{D}, \texttt{E}\}$).

Many works have been done on mining constrained itemsets. One of the well-studied constraint is the frequency constraint. Although an exponential search space is concerned, it is known that frequent itemsets can be computed in real-life large datasets thanks to the Apriori trick that drastically reduces the search space [1]. The Apriori algorithm can be used for mining tasks such as the one presented in example 1: first compute all frequent itemsets using Apriori, then remove itemsets that do not contain attribute `A`. We call this strategy "generate and test". However, this strategy can easily be improved. The constraint can be "pushed" inside the Apriori algorithm, i.e., perform Apriori only on itemsets that contain attribute `A` (counting the frequency of itemsets that do not contain `A` can be avoided). In many cases, pushing constraints is very effective and many papers present ways to "push" different kinds of constraint [15, 9, 7]. However, none of them discuss the question: is it always profitable to "push" constraints?

We show in this paper that the answer is negative. Using a generic algorithm, we discuss the effective use of arbitrary constraints i.e., how they can be pushed into the discovery algorithm for optimization purposes. If it is rather easy for the so-called anti-monotone constraints (e.g., the frequency constraint), it becomes harder for constraints that are not anti-monotone. Pushing these latter constraints can avoid expensive anti-monotone constraint checking but might lead to less effective pruning phases. Among others, we discuss the interesting case of monotone constraints (negations of anti-monotone constraints).

The paper is organized as follows. Section 2 provides a simple formalization of the kind of inductive query we have to process. In Section 3, using an abstract presentation of the classical Apriori algorithm, we introduce the problem of frequent itemset discovery under constraints and we study the simple case of anti-monotone constraints. In section 3, we use a simple taxonomy of constraints and discuss evaluation schemes of inductive queries. Section 4 revisits the Close algorithm when considering its specific pruning criterion as a constraint checking step. Finally, section 6 concludes.

[1] For instance, assuming the popular basket analysis task, `Items` denotes a collection of products that belong or not to commercial transactions.

2 Inductive Databases and Inductive Queries

We consider the formalization of inductive databases [4] and the concept of inductive query in our context.

Definition 1 (schema and instance). *The schema of an inductive database is a pair* $\mathcal{R} = (\mathbf{R}, (\mathcal{L}, \mathcal{E}, \mathcal{V}))$*, where* $\mathbf{R}$ *is a database schema,* $\mathcal{L}$ *is a countable collection of patterns,* $\mathcal{V}$ *is a set of* result values*, and* $\mathcal{E}$ *is the* evaluation function *that characterizes patterns. Given a database* $\mathbf{r}$ *over* $\mathbf{R}$ *and a pattern* $\theta \in \mathcal{L}$*, this function maps* $(\mathbf{r}, \theta)$ *to an element of* $\mathcal{V}$*. An instance* $(\mathbf{r}, s)$ *over* $\mathcal{R}$ *consists of a database* $\mathbf{r}$ *over the schema* $\mathbf{R}$ *and a subset* $s \subseteq \mathcal{L}$*.*

A typical KDD process operates on both of the components of an inductive database. Queries that concerns only the pattern part, called hereafter *inductive queries*, specify mining tasks.

Definition 2 (inductive query). *Given an inductive database instance* $(\mathbf{r}, s)$ *whose schema is* $(\mathbf{R}, (\mathcal{L}, \mathcal{E}, \mathcal{V}))$*, an inductive query is denoted as* $\sigma_{\mathcal{C}}(s)$ *and specifies sentences from* s *which are interesting.* $\mathcal{C}$ *is a conjunction of constraints that must be fulfilled by the desired patterns. Checking some of the conjuncts may need for the evaluation of* $\mathcal{E}$ *on* $\mathbf{r}$ *and involve data scans.*

Example 2. Assume the minable view is `trans(Tid,Items)` i.e., a typical schema of data for basket analysis. Figure 1 provides a toy dataset under a boolean matrix format. For instance, if `trans(2,A)` and `trans(2,C)` define the transaction 2, row 2 contains true for columns `A` and `C` and false for column `B`, `D` and `E`.

In this paper, we always consider the database schema of example 1. In this context we can now define the *itemset mining task*.

Definition 3 (itemset mining task). *An itemset denotes a subset of* `Items`*. Let* $\mathcal{F}(S, \mathbf{r})$*, the frequency of* S*, denotes the percentage of transactions that involve each attribute in* S*. An itemset* S *is* γ*-frequent in* $\mathbf{r}$ *if* $\mathcal{F}(S, \mathbf{r}) \geq \gamma$*. This constraint is denoted by* $\mathcal{C}_{freq}$*. The pattern part of the associated inductive database is* $(\mathcal{L}, \mathcal{E}, [0, 1])$ *where* $\mathcal{L} = 2^{\texttt{Items}}$ *and* $\mathcal{E}$ *returns the itemset frequency. The constrained itemset mining task (itemset mining task, for short) is the evaluation of* $\sigma_{\mathcal{C}}(2^{\texttt{Items}})$ *where* $\mathcal{C}$ *is a conjunction of constraints. Mining* frequent *constrained itemsets means that* $\mathcal{C}$ *contains* $\mathcal{C}_{freq}$*.*

$\mathbf{r} =$

TID	A	B	C	D	E
1	1	1	1	1	1
2	1	0	1	0	0
3	1	1	1	1	0
4	0	1	1	0	0
5	1	1	1	0	0
6	0	0	0	0	1

Itemset	Frequency
{A}	0.67
{B}	0.67
{C}	0.83
{A,B}	0.5
{A,C}	0.67
{B,C}	0.67
{C,D}	0.33
{A,C,D}	0.33

Fig. 1. A binary dataset $\mathbf{r}$ and the frequencies of some itemsets.

3 Formalizing the Use of Constraints

3.1 Preliminaries

An itemset S satisfies a constraint $\mathcal{C}$ iff $\mathcal{C}(S)$ evaluates to true. If $\mathcal{C}$ is a constraint, let $SAT_{\mathcal{C}}(\texttt{Items})$ denotes the collection $\{S \subseteq \texttt{Items}, S \text{ satisfies } \mathcal{C}\}$, i.e., $\sigma_{\mathcal{C}}(2^{\texttt{Items}})$. To avoid confusion, the constraints are sometimes defined by "$\equiv$" instead of "$=$" (e.g., $\mathcal{C}(S) \equiv |S| = 10$ is a constraint whose value is true iff the size of the itemset S is 10). $\texttt{Items}_k$ denotes the collection $\{S \subseteq \texttt{Items}, |S| = \text{k}\}$ of the itemsets of size k.

Example 3. Consider the dataset of figure 1 where $\texttt{Items}=\{\texttt{A},\texttt{B},\texttt{C},\texttt{D},\texttt{E}\}$. If the frequency constraint $\mathcal{C}_{freq}$ specifies that an itemset must be 0.6-frequent, the inductive query $\sigma_{\mathcal{C}_{freq}}(2^{\texttt{Items}})$ returns $\{\texttt{A},\texttt{B},\texttt{C},\texttt{AC},\texttt{BC}\}$. Assume that $\mathcal{C}_{size}(S) \equiv |S| \leq 3$ and $\mathcal{C}_{miss}(S) \equiv \{\texttt{B},\texttt{E}\} \cap S = \emptyset$. The query $\sigma_{\mathcal{C}_{size} \wedge \mathcal{C}_{miss}}(2^{\texttt{Items}})$ returns $\{\texttt{A},\texttt{C},\texttt{D},\texttt{AC},\texttt{AD},\texttt{CD},\texttt{ACD}\}$ while the query $\sigma_{\mathcal{C}_{freq} \wedge \mathcal{C}_{size} \wedge \mathcal{C}_{miss}}(2^{\texttt{Items}})$ returns $\{\texttt{A},\texttt{C},\texttt{AC}\}$. Notice that we use a string notation (e.g., $\texttt{AC}$) to denote sets of attributes.

An *anti-monotone* constraint is a constraint $\mathcal{C}$ such that for all itemsets S, S': $(S' \subseteq S \wedge S \text{ satisfies } \mathcal{C}) \Rightarrow S' \text{ satisfies } \mathcal{C}$.

Example 4. $\mathcal{C}_{freq}$, $\mathcal{C}(S) \equiv \texttt{A} \notin S$, $S \subseteq \{\texttt{A},\texttt{B},\texttt{C}\}$ and $S \cap \{\texttt{A},\texttt{B},\texttt{C}\}$ are examples of anti-monotone constraints. Many other anti-monotone constraints are presented in [13].

It is clear that a disjunction or a conjunction of anti-monotone constraints is an anti-monotone constraint. A useful concept that enables reasoning on frequent itemset mining algorithms is the concept of (negative or positive) border [12]. If $\mathcal{C}_{am}$ denotes an anti-monotone constraint and $T \subseteq 2^{\texttt{Items}}$, $\mathcal{B}d^{-}_{\mathcal{C}_{am}}(T)$ is the collection of the minimal (w.r.t. the set inclusion) itemsets of T that do not satisfy $\mathcal{C}_{am}$. $\mathcal{B}d^{+}_{\mathcal{C}_{am}}(T)$ is the collection of the maximal (w.r.t. the set inclusion) itemsets of T that satisfy $\mathcal{C}_{am}$. Finally, we assume $\mathcal{B}d^{+}_{\mathcal{C}_{am}} = \mathcal{B}d^{+}_{\mathcal{C}_{am}}(2^{\texttt{Items}})$ and $\mathcal{B}d^{-}_{\mathcal{C}_{am}} = \mathcal{B}d^{-}_{\mathcal{C}_{am}}(2^{\texttt{Items}})$.

Given a constraint $\mathcal{C}$, our problem is now to find an algorithm that performs the itemset mining task (i.e., that computes $\sigma_{\mathcal{C}}(\texttt{Items}) = SAT_{\mathcal{C}}(\texttt{Items})$). Given an algorithm $\mathcal{A}$, we call $\texttt{Test}_{\mathcal{A}}(\mathcal{C})$ the set of the itemsets that are tested against the constraint $\mathcal{C}$ by $\mathcal{A}$. For example, it is well known that, in the case of the Apriori algorithm, $\texttt{Test}_{apriori}(\mathcal{C}_{freq}) = SAT_{\mathcal{C}_{freq}} \cup \mathcal{B}d^{-}_{\mathcal{C}_{freq}}$ [12]. The selectivity of a constraint in an algorithm $\mathcal{A}$ is the number of itemsets S in $\texttt{Test}_{\mathcal{A}}(\mathcal{C})$ such that $\mathcal{C}(S)$ is false (i.e., the number of rejected itemsets).

3.2 A Simple Case: Testing Anti-monotone Constraints

We consider an abstract definition of the Apriori algorithm [1] to support our discussion on the effective use of constraints. This algorithm performs the frequent itemset mining task when $\mathcal{C} = \mathcal{C}_{freq}$.

APRIORI **algorithm**

1. $C_1 := \texttt{Items}_1$
2. k := 1
3. **while** $C_k \neq \emptyset$ **do**
4. Phase 1 - frequency constraint is enforced - it needs a data scan
   $\mathcal{L}_k := SAT_{\mathcal{C}_{freq}} \cap C_k$
5. Phase 2 - candidate generation for level k+1
   $C^g_{k+1} := \texttt{generate}_{apriori}(\mathcal{L}_k)$
6. Phase 3 - candidate safe pruning
   $C_{k+1} := \texttt{safe-pruning-on}(C^g_{k+1})$
7. $k := k + 1$

   od
8. **output** $\bigcup_{i=1}^{k-1} \mathcal{L}_i$

In the classical APRIORI algorithm, $\texttt{generate}_{apriori}(\mathcal{L}_k)$ provides the candidates by fusion of two elements from $\mathcal{L}_k$ that share the same $k-1$ first items: $\texttt{generate}_{apriori}(\mathcal{L}_k)$=$\{A \cup B$, where $A, B \in \mathcal{L}_k$, A and B share the $k-1$ first items (in lexicographic order)$\}$. Furthermore, in its standard presentation, phase 2 and 3 are merged. $\texttt{safe-pruning-on}(C^g_{k+1})$ eliminates the candidates for which a subset of length k is not frequent. This can be justified by the APRIORI trick: if S is not frequent, every superset of S is not frequent.

Example 5. If $\mathcal{L}_2 = \{\texttt{AB}, \texttt{AC}, \texttt{BC}, \texttt{AD}, \texttt{CD}\}$, phase 2 provides the collection of candidates $C^g_3 = \{\texttt{ABC}, \texttt{ABD}, \texttt{ACD}\}$ (BCD is not generated, $\texttt{generate}_{apriori}$ already performs some pruning). Phase 3 provides $C_3 = \{\texttt{ABC}, \texttt{ACD}\}$ since $\texttt{BD} \notin \mathcal{L}_2$.

This trick uses only the anti-monotonicity of $\mathcal{C}_{freq}$ and can be generalized. Let $\mathcal{C} = \mathcal{C}_{am}$ be an anti-monotone constraint eventually including $\mathcal{C}_{freq}$: if S does not satisfy $\mathcal{C}_{am}$, every superset of S does not satisfy $\mathcal{C}_{am}$. It is therefore straightforward to use the APRIORI algorithm with any anti-monotone constraint. We are now able to find a strategy to perform the itemset mining task for a constraint $\mathcal{C} = \mathcal{C}_{am} \wedge \mathcal{C}_o$ that is the conjunction of an anti-monotone constraint and another constraint.

Definition 4 (generate and test strategy). *Assume we want to perform the itemset mining task with the constraint $\mathcal{C} = \mathcal{C}_{am} \wedge \mathcal{C}_o$. The generate and test strategy consist of two steps. First, compute $SAT_{\mathcal{C}_{am}}$ with* APRIORI *algorithm. Second, test each itemset from the result of the first step against the constraint $\mathcal{C}_o$.*

Assume now that $\mathcal{C}$ is the conjunction of two anti-monotone constraints $\mathcal{C}_{am}$ and $\mathcal{C}'_{am}$. The optimization problem is to know whether it is more efficient (strategy push) to use the APRIORI algorithm with the constraint $\mathcal{C} = \mathcal{C}_{am} \wedge \mathcal{C}'_{am}$ (to evaluate $\mathcal{C}(S)$, $\mathcal{C}_{am}(S)$ is evaluated and then $\mathcal{C}'_{am}(S)$ *if* $\mathcal{C}_{am}(S)$ *is true*), or (strategy g&t) to use the "generate and test" strategy, i.e., to first generate $SAT_{\mathcal{C}_{am}}$

with APRIORI and then to test each $S \in SAT_{\mathcal{C}_{am}}$ against the constraint $\mathcal{C}'_{am}$. Under these hypotheses, we have the following theorem.

Theorem 1. $\texttt{Test}_{\texttt{push}}(\mathcal{C}_{am}) \subseteq \texttt{Test}_{\texttt{g\&t}}(\mathcal{C}_{am})$ *and* $\texttt{Test}_{\texttt{push}}(\mathcal{C}'_{am}) \subseteq \texttt{Test}_{\texttt{g\&t}}(\mathcal{C}'_{am})$, *where* $\texttt{Test}_{\texttt{push}}$ *corresponds to strategy* `push` *and* $\texttt{Test}_{\texttt{g\&t}}$ *to strategy* `g&t`.

Proof of theorem 1.
Strategy `g&t`: it is the APRIORI algorithm with $\mathcal{C}_{am}(S)$ therefore $\texttt{Test}_{\texttt{g\&t}}(\mathcal{C}_{am}) = SAT_{\mathcal{C}_{am}} \cup \mathcal{B}d^-_{\mathcal{C}_{am}}$ and $\mathcal{C}'_{am}$ is tested on the result of this algorithm, $\texttt{Test}_{\texttt{g\&t}}(\mathcal{C}'_{am}) = SAT_{\mathcal{C}_{am}}$.
Strategy `push`: due to the order in which $\mathcal{C}_{am}$ and $\mathcal{C}'_{am}$ are tested, $\texttt{Test}_{\texttt{push}}(\mathcal{C}_{am}) = SAT_{\mathcal{C}_{am} \wedge \mathcal{C}'_{am}} \cup \mathcal{B}d^-_{\mathcal{C}_{am} \wedge \mathcal{C}'_{am}}$ and $\texttt{Test}_{\texttt{push}}(\mathcal{C}'_{am}) = SAT_{\mathcal{C}_{am} \wedge \mathcal{C}'_{am}} \cup (\mathcal{B}d^-_{\mathcal{C}_{am} \wedge \mathcal{C}'_{am}} \cap SAT_{\mathcal{C}_{am}})$.
The second statement of the theorem follows. For the first one, we must prove that $SAT_{\mathcal{C}_{am} \wedge \mathcal{C}'_{am}} \cup \mathcal{B}d^-_{\mathcal{C}_{am} \wedge \mathcal{C}'_{am}} \subseteq SAT_{\mathcal{C}_{am}} \cup \mathcal{B}d^-_{\mathcal{C}_{am}}$: It is clear that $SAT_{\mathcal{C}_{am} \wedge \mathcal{C}'_{am}} \subseteq SAT_{\mathcal{C}_{am}}$. Let $T \in \mathcal{B}d^-_{\mathcal{C}_{am} \wedge \mathcal{C}'_{am}}$. $\forall S \subset T, \mathcal{C}_{am}(S) \wedge \mathcal{C}'_{am}(S)$ is true, so $\mathcal{C}_{am}(S)$ is true. Therefore if $\mathcal{C}_{am}(T)$ is false, $T \in \mathcal{B}d^-_{\mathcal{C}_{am}}$. If $\mathcal{C}_{am}(T)$ is true, $T \in SAT_{\mathcal{C}_{am}}$. In either case, $T \in SAT_{\mathcal{C}_{am}} \cup \mathcal{B}d^-_{\mathcal{C}_{am}}$ and the theorem is true.

This theorem states that strategy `push` leads to fewer constraint checking than strategy `g&t`. So, it is always profitable to push anti-monotone constraints (the generate and test strategy is less efficient for anti-monotone constraints).

4 Optimizing Constraint Checking

4.1 Testing non Anti-monotone Constraints

Assume $\mathcal{C}$ is a conjunction of atomic constraints $\mathcal{C}_1 \wedge \ldots \wedge \mathcal{C}_n$, some of which are anti-monotone (e.g., $\mathcal{C}_{freq}$) and some others are not. Let the conjunction of all anti-monotone constraints from $\mathcal{C}_1 \wedge \ldots \wedge \mathcal{C}_n$ be denoted $\mathcal{C}_{am}$ and the conjunction of other constraints be denoted $\mathcal{C}_o$.

In the previous section, we show that it is interesting to push every anti-monotone constraint checking during the search space exploration. Therefore, it is natural to try to push constraints that are not anti-monotone.

The motivation is that testing some anti-monotone constraints is very expensive, e.g., those like $\mathcal{C}_{freq}$ that need to scan the data. One solution is to remove as soon as possible some candidates by pushing non anti-monotone constraints during the search space exploration. A particular case is when it is possible to push the constraint at the candidate generation step: for example, using the constraint $\texttt{A} \in S$ avoids the generation of the candidates that do not contain `A`, and therefore minimize the constraint testing (the frequencies of the itemsets that do not contain `A` do not have to be computed).

However, pushing non anti-monotone constraints leads to less effective pruning. Pruning, in the case of anti-monotone constraints, is based on the fact that, if an itemset S does not satisfy the anti-monotone constraint $\mathcal{C}_{am}$, then every superset of S do not satisfy it either. However, if S has been removed from the

set of the candidates before the test of $\mathcal{C}_{am}$ because it does not verify some non anti-monotone constraint, $\mathcal{C}_{am}(S)$ is unknown. Then, it is not possible to prune the supersets of S.

The tradeoff is as follows: when a non anti-monotone constraint is pushed, it might save tests on more costly anti-monotone constraints. However, the results of these tests could have lead to more effective pruning. The following example shows that the "generate and test" strategy is sometimes more efficient.

Example 6. Assume the constraint $\mathcal{C}(S) \equiv |S| = 10 \wedge \mathcal{C}_{freq}(S)$, i.e., S contains exactly 10 items and S is frequent. If the constraint $|S| = 10$ is pushed into the candidate generation step, no candidate of size lower than 10 is generated. Every candidate of size 10 is generated and its frequency is tested in one database pass. It is clear that no pruning is possible. This leads to $\binom{n}{10}$ candidates and, as soon as n is large, this turns to be intractable for any frequency threshold.

A "generate and test" strategy computes every frequent itemset and needs several passes over the database. Then all itemsets whose size is not 10 are removed. This strategy remains tractable for a reasonable frequency threshold even for a large n.

This example is one of the most important of the paper. It shows that pushing non anti-monotone constraints can be a mistake. Previous works on this subject never point out this tradeoff. This motivates our general study about pushing non anti-monotone constraints.

4.2 A Generic Algorithm

If one wants to push an non anti-monotone constraint in the loop of the Apriori algorithm, there are several possibilities (e.g., the constraint can be pushed between phase 1 and phase 2 or between phase 2 and phase 3, etc.). To study these possibilities, we propose a generic algorithm $\mathcal{G}$ that generalizes Apriori and that allows to push non anti-monotone constraints at several places in the loop. In this algorithm and in the whole section, k denotes the size of the itemsets at the current level, $\mathcal{L}_k$ denotes the set of itemsets of size k that are generated in step 4 and that pass all the tests of steps 5 through 9.

Steps 3 to 10 define the loop of the algorithm. $\mathcal{C}_a, \mathcal{C}_b, \mathcal{C}_c$ and $\mathcal{C}_d$ are conjunctions of atomic constraints from $\mathcal{C} = \mathcal{C}_1 \wedge \ldots \wedge \mathcal{C}_n$. $\mathcal{C}_{dbs}$ is the conjunction of the atomic constraints that need a database pass to be tested (e.g., $\mathcal{C}_{freq}$). There are several steps where an atomic constraint $\mathcal{C}_i$ can be tested. First, it can be pushed at the candidate generation step (step 4) if it is possible to have a generation procedure that makes use of the constraint (e.g., $\mathcal{C}_i(S) \equiv \mathtt{A} \in S$). This is discussed in the paragraph about candidate generation. Second, it is possible to test it on steps 5, 7, 8, 9 and 12. Step 12 corresponds to a "generate and test" strategy: the constraint is not pushed into the search space exploration.

The safe pruning step (step 6) can be seen as a constraint checking step too. Pruning means exactly removing itemsets S that cannot verify the anti-monotone

Algorithm $\mathcal{G}$

1. $k := 1$
2. $\mathcal{L}_0 := \{\emptyset\}$
3. **repeat**
   # Candidate generation for level k
4. $C_k^0 := \texttt{generate}(\mathcal{L}_{k-1})$
5. $C_k^1 := C_k^0 \cap SAT_{\mathcal{C}_a}$ # Test $\mathcal{C}_a$
6. # Safe pruning of C_k^1 using anti-monotone constraints
   $C_k^2 := \texttt{safe-pruning-on}(C_k^1)$
7. $C_k^3 := C_k^2 \cap SAT_{\mathcal{C}_b}$ # Test $\mathcal{C}_b$
   #Test constraints which need a database scan
8. $C_k^4 := C_k^3 \cap SAT_{\mathcal{C}_{dbs}}$
9. $\mathcal{L}_k := C_k^4 \cap SAT_{\mathcal{C}_c}$ # Test $\mathcal{C}_c$
10. $k := k + 1$
11. **until** there is no more candidates
    # Test constraint $\mathcal{C}_d$ on extracted itemsets
12. **return** $\cup_{i=1}^{k-1} \mathcal{L}_i \cap SAT_{\mathcal{C}_d}$

constraint $\mathcal{C}_{am}$ because we already know (from a previous iteration) that a subset of S does not verify it. Implementing this constraint is discussed in a specific paragraph. Step 6 can be rewritten $C_k^2 := C_k^1 \cap SAT_{\mathcal{C}_{prune}}$, where $\mathcal{C}_{prune}(S)$ is true iff S cannot be pruned.

Finally, each of the steps of the generic algorithm can be seen as a constraint checking step. This means that we can rewrite the loop of the algorithm $\mathcal{G}$ as:

repeat
$C_k^0 := \texttt{generate}(\mathcal{L}_{k-1})$
$C_k^1 := C_k^0 \cap SAT_{\mathcal{C}'_1}$ # Test $\mathcal{C}'_1$
$C_k^2 := C_k^1 \cap SAT_{\mathcal{C}'_2}$ # Test $\mathcal{C}'_2$
...
$\mathcal{L}_k := C_k^{m-1} \cap SAT_{\mathcal{C}'_m}$ # Test $\mathcal{C}'_m$
$k := k + 1$
until there is no more candidates

where each atomic constraint $\mathcal{C}'_i$, $1 \leq i \leq m$, is either one of the $\mathcal{C}_j$, $1 \leq j \leq n$, or the pruning constraint $\mathcal{C}_{prune}$ (p denotes the index such that $\mathcal{C}_{prune} = \mathcal{C}'_p$). Reciprocally, each of the atomic constraints $\mathcal{C}_j$, $1 \leq j \leq n$ is either one of the $\mathcal{C}'_i$, $1 \leq i \leq m$, or in the compound constraint $\mathcal{C}_d$ (and outside the loop).

It is straightforward to see that the algorithm $\mathcal{G}$ is correct (i.e., it returns a subset of $SAT_{\mathcal{C}}$). The completeness of $\mathcal{G}$ depends of the generation and pruning steps. The generation step must be complete, i.e., the candidate set C_k^0 must be a superset of $SAT_{\mathcal{C}} \cap \texttt{Items}_k$. The pruning step must be correct, i.e., it must not remove any itemset that is in $SAT_{\mathcal{C}}$.

Theorem 2. *Provided that the generation step is complete and the pruning step is correct, the generic algorithm $\mathcal{G}$ is correct and complete, i.e., it returns $SAT_{\mathcal{C}}$.*

Ordering the Constraints in the Loop Pushing an atomic constraint $\mathcal{C}'$ from step 12 to the last step of the loop of the generic algorithm $\mathcal{G}$ will not change $\texttt{Test}(\mathcal{C}')$ or $\texttt{Test}(\mathcal{C}'_i)$ for any $1 \leq i \leq m$. However, once $\mathcal{C}'$ is inside the loop, the constraints can be reordered.

Two parameters influence the search for a good ordering: the selectivity of the constraints and their evaluation cost. Selective constraints avoid a lot of checking steps for the constraints that follow. However, it is generally not possible to have a clear idea of the selectivity of a constraint.

Example 7. Consider the constraint $\mathcal{C}(S) \equiv \mathcal{C}_{freq}(S) \wedge S \subseteq \{\texttt{A}, \texttt{C}, \texttt{E}\}$. If every itemset is frequent, $S \subseteq \{\texttt{A}, \texttt{C}, \texttt{E}\}$ has a high selectivity. On the other hand, if `A`, `C` and `E` are the only frequent attributes, its selectivity is nil.

However, it is possible to provide a few remarks about constraint ordering.

- If a non anti-monotone constraint is tested before an anti-monotone one, it implies less pruning. This is valuable only if it avoids checking some other more expensive constraints (e.g., frequency constraint checking).
- In the generic algorithm $\mathcal{G}$, there is only one step (step 8) during which a database pass is performed. It is theoretically possible to have several such steps (there can be several constraints that need data scans). However, several steps with a database access mean several database passes. For very large datasets, it can be highly desirable to have at most one database pass per level.
- If an anti-monotone constraint is tested before the pruning constraint $\mathcal{C}_{prune}$, it does not take any profit from the pruning phase.

Candidate Generation Naive generation leads to $\binom{n}{k}$ candidates and is clearly intractable. It is mandatory to design a candidate generation algorithm that is complete (a superset of $SAT_{\mathcal{C}} \cap \texttt{Items}_k$ has to be generated) and effective ($\cup_{k \geq 1} C_k^0$ must be as small as possible).

In the general case, $\mathcal{C} = \mathcal{C}_{am} \wedge \mathcal{C}_o$ and the only information available is that there is no need to generate itemsets that are supersets of itemsets that do not verify $\mathcal{C}_{am}$. Let R_k denote the set of itemsets S of size k for which we have found that $\mathcal{C}_{am}(S)$ is false (R_k is a subset of $C_k^0 - \mathcal{L}_k$), $N_k = \cup_{i \leq k} R_i$ and $N_k^+ = \{S \subseteq \texttt{Items}, \exists S' \in N_k \text{ s.t. } S' \subseteq S\}$. We have the following theorem.

Theorem 3. *The candidate generation step is complete with any constraint $\mathcal{C} = \mathcal{C}_{am} \wedge \mathcal{C}_o$ only if C_{k+1}^0 is a superset of $\texttt{Items}_{k+1} - N_k^+$ and $C_0^0 = \texttt{Items}$.*

As a consequence, the classical candidate generation step of the Apriori algorithm, $\texttt{generate}_{apriori}$, does not work for general constraints (because it makes the assumption that all itemsets that verify $\mathcal{C}_{am}$ are in $\mathcal{L}_k$).

We do not know an efficient algorithm to enumerate $\texttt{Items}_{k+1} - N_k^+$ (without the obvious intractable generation of $\texttt{Items}_{k+1}$). Such an algorithm would be very valuable: pruning step would no longer be necessary.

Pruning The goal of the pruning step is to remove from the set of candidates C_{k+1}^{p-1} the itemsets of N_k^+. The classical APRIORI candidate generation ($\texttt{generate}_{apriori}$) generates a superset of $\texttt{Items}_{k+1} - N_k^+$. A subsequent pruning step is necessary to remove all itemsets of N_k^+ from the set of candidates C_{k+1}. The APRIORI algorithm uses the following pruning algorithm:

Pruning: algorithm $\mathcal{P}$
for all $S \in C_{k+1}$ and **for all** $S' \subset S$ such that $|S'| = k$
do if $S' \notin \mathcal{L}_k$ **then** delete S from C_{k+1} **od**

Remember that $\mathcal{L}_k$ is the set of itemsets of size k that verify all the constraints pushed in the loop (if all the constraints are pushed, $\mathcal{L}_k = SAT_{\mathcal{C}} \cap \texttt{Items}_k$). The important property for a pruning algorithm is its correctness. It ensures that no itemset which verifies the constraints is pruned. The following proposition is a straightforward characterization of the itemsets that can be safely pruned.

Proposition 1. *A pruning algorithm is correct if at level $k+1$, S is pruned only if $S \in N_k^+$.*

The pruning algorithm $\mathcal{P}$ is correct for anti-monotone constraints only, i.e., if $\mathcal{C}$ is an anti-monotone constraint. In the general case, $\mathcal{C} = \mathcal{C}_{am} \wedge \mathcal{C}_o$ and $\mathcal{C}_o$ is not anti-monotone. It can be so that $S' \notin \mathcal{L}_k$ because $\mathcal{C}_o(S')$ is false while $\mathcal{C}_{am}(S')$ is true. In such a case, the algorithm would prune $S \supset S'$ while $\mathcal{C}_o(S) \wedge \mathcal{C}_{am}(S)$ is true (incorrectness). Let us propose a correct pruning algorithm.

Pruning: algorithm $\mathcal{P}'$
for all $S \in C_{k+1}$ and **for all** $S' \subset S$ such that $|S'| = k$
do if $S' \notin \mathcal{L}_k$ **and** $\mathcal{C}_o(S')$ **then** delete S from C_{k+1} **od**

Proposition 2. *$\mathcal{P}'$ is correct for all types of constraints $\mathcal{C}$*

If $S' \notin \mathcal{L}_k$ and $\mathcal{C}(S')$ is true then $\mathcal{C}_{am}(S')$ is false. It follows that, due to the anti-monotonicity, if $S' \subset S$ then $\mathcal{C}_{am}(S)$ is false and the candidate S can be pruned safely. We discuss now the completeness of the algorithm $\mathcal{P}'$.

Definition 5 (complete pruning algorithm). *A pruning algorithm is complete if at level $k+1$, $S \in N_k^+ \Rightarrow S$ is pruned.*

Proposition 3. *$\mathcal{P}'$ is not complete.*

Example 8. In this example, $\texttt{Items} = \{\texttt{A}, \texttt{B}, \texttt{C}, \texttt{D}\}$. Assume all itemsets are frequent except CD and its supersets (BCD, ACD and ABCD). Assume the constraint is $\mathcal{C} = \mathcal{C}_{freq} \wedge \mathcal{C}_o$ where $\mathcal{C}_o(S) \equiv (|S| \geq 3 \Rightarrow \{\texttt{A}, \texttt{B}\} \subset \texttt{S})$, i.e., the itemsets that do

not satisfy $\mathcal{C}_o$ are ACD and BCD. At level 3, $\mathcal{L}_3 = \{\texttt{ABC}, \texttt{BCD}\}$. At level 4, ABCD is generated (from ABC and BCD) and is not pruned by $\mathcal{P}'$ since for all $S' \subset \texttt{ABCD}$ such that $|S'| = 3$ and $S' \notin \mathcal{L}_3$, $\mathcal{C}_o(S')$ is false. However, at level 2 the algorithm finds that CD is not frequent, so ABCD should be pruned since it is a superset of CD. $\mathcal{P}'$ is therefore not complete.

Another problem with the algorithm $\mathcal{P}'$ is that it can be costly to test $\mathcal{C}_o(S')$ for each subset of the itemsets of C_{k+1}.

A correct and complete pruning algorithm needs to remove each itemset of C_{k+1} that is the superset of an itemset from N_k. This can be done easily if itemsets of N_k are stored, however this can be expensive.

4.3 Testing Monotone Constraints

In previous sections, we have seen that it is difficult to tackle the problem of the candidate generation step with general constraints. Anti-monotone constraints provide information on itemsets that cannot satisfy the constraints. We need also information to find a superset of the itemsets that verify the constraints to have a correct and efficient candidate generation algorithm. It seems natural to consider the negation of anti-monotone constraints.

In this section, we focus on conjunctions of anti-monotone and monotone constraints, i.e., $\mathcal{C} = \mathcal{C}_{am} \wedge \mathcal{C}_m$. A monotone constraint $\mathcal{C}_m$ is simply the negation of an anti-monotone constraint. If $\mathcal{C}_m$ is a monotone constraint, we note $\mathcal{C}_m(S) = \neg\mathcal{C}'_{am}(S)$. The main property of a monotone constraint is: $S \subseteq \texttt{Items}$, $\mathcal{C}_m(S)$ is true $\Rightarrow \forall S' \supset S$, $\mathcal{C}_m(S')$ is true.

Example 9. Continuing our running example, $\{\texttt{A},\texttt{B},\texttt{C},\texttt{D}\} \subset S$ and $S \cap \{\texttt{A},\texttt{B},\texttt{C}\} \neq \emptyset$ are monotone constraints on S.

Anti-monotone constraints give us the possibility to prune some candidates. Monotone constraints provide a candidate generation strategy:

Proposition 4. *If $\mathcal{C}(S) = \mathcal{C}_{am}(S) \wedge \neg\mathcal{C}'_{am}(S)$ is true then exactly one of these properties is true:*

- $S \in \mathcal{B}d^-_{\mathcal{C}'_{am}}$
- $\exists S' \subset S, |S'| = |S| - 1$ *and* $\mathcal{C}(S')$ *is true.*

This property provides a superset of the candidate set: if $\texttt{generate}_1(\mathcal{L}_k) = \{A \cup B$, where $A \in \mathcal{L}_k$, $B \in F_1$ and $B \not\subset A\}$, (F_1 is $SAT_{\mathcal{C}_{am}} \cap \texttt{Items}_1$) then $C_{k+1} = \texttt{generate}_1(\mathcal{L}_k) \cup (\mathcal{B}d^-_{\mathcal{C}'_{am}} \cap \texttt{Items}_{k+1})$ is a correct candidate set for level $k+1$. The remaining problem is the generation of $\mathcal{B}d^-_{\mathcal{C}'_{am}}$. It is not a trivial problem. However several methods are available, e.g., one can use Apriori with constraint $\mathcal{C}'_{am}$ to find $\mathcal{B}d^-_{\mathcal{C}'_{am}}$.

Let us now consider another generating procedure, close to the Apriori's original one: $\texttt{generate}_2(\mathcal{L}_k) = \{A \cup B$, where $A, B \in \mathcal{L}_k\}$. A naive algorithm to compute $\texttt{generate}_2$ will produce many duplicates (see [7]).

Theorem 4. *Assume* $ms = \mathit{Max}_{S \in \mathcal{B}d^-_{\mathcal{C}'_{am}}} |S|$. *If* $C_1 = \mathcal{B}d^-_{\mathcal{C}'_{am}} \cap \mathtt{Items}_1$ *and, for* $k \geq 1$, *the candidate set* C_{k+1} *is defined by:*
if $k < ms$, $C_{k+1} = \mathtt{generate}_1(\mathcal{L}_k) \cup (\mathcal{B}d^-_{\mathcal{C}'_{am}} \cap \mathtt{Items}_{k+1})$;
if $k = ms$, $C_{k+1} = \mathtt{generate}_1(\mathcal{L}_k)$;
if $k > ms$, $C_{k+1} = \mathtt{generate}_2(\mathcal{L}_k)$;
then this candidate generation procedure is complete and ensures that every candidate itemset verifies $\neg\, \mathcal{C}'_{am} = \mathcal{C}_m$.

The generic algorithm with the generation step given by this theorem is therefore correct and complete. The fact that every candidate itemset verify $\mathcal{C}_m$ makes useless any verification of this constraint after the candidate generation step.

In the previous section we saw that pruning algorithm $\mathcal{P}'$ is correct.

Proposition 5. *If* $\mathcal{C} = \mathcal{C}_{am} \wedge \mathcal{C}_m$ *then pruning algorithm* $\mathcal{P}'$ *is complete.*

5 Revisiting the Close Algorithm

It is now interesting to revisit the algorithm Close [14] [3]. This algorithm make the frequent itemset discovery possible in highly-correlated data where Apriori is generally intractable. We can see this algorithm as an instantiation of the generic algorithm with an anti-monotone constraint $\mathcal{C}_{freq} \wedge \mathcal{C}_{Close}$ and with the same candidate generation and pruning algorithms as that of Apriori ($\mathtt{generate}_{apriori}$ candidate generation algorithm and pruning algorithm $\mathcal{P}$).

Definition 6 (new constraint for Close). $\mathcal{C}_{Close}(S) \equiv S' \subset S \Rightarrow S \not\subseteq \mathtt{closure}(S')$ *where* $\mathtt{closure}(S)$ *is the maximal (for the set inclusion) superset of* S *which has the same frequency as* S.

An important property of the Close algorithm is its completeness: it is possible to compute $SAT_{\mathcal{C}_{freq}}$ and the frequency of each frequent itemset knowing Close's output. Indeed, Close provides also the closures of the itemsets from $SAT_{\mathcal{C}_{Close} \wedge \mathcal{C}_{freq}}$. The demonstration of completeness is done in [14].

Example 10. Let us find $\mathtt{closure}(\mathtt{AB})$ on our running example. Items A and B are simultaneously present in rows 1, 3 and 5. We notice that item C is the only other item that is also present in these three rows, thus $\mathtt{closure}(\mathtt{AB}) = \mathtt{ABC}$. We also have $\mathtt{closure}(\mathtt{A}) = \mathtt{AC}$ and $\mathtt{closure}(\mathtt{B}) = \mathtt{BC}$, so $\mathtt{AB} \not\subseteq \mathtt{closure}(\mathtt{A})$ and $\mathtt{AB} \not\subseteq \mathtt{closure}(\mathtt{B})$. Therefore $\mathcal{C}_{Close}(\mathtt{AB})$ is true. If the frequency threshold is $\gamma = 1/3$, the inductive query $\sigma_{\mathcal{C}_{freq} \wedge \mathcal{C}_{Close}}(2^{\mathtt{Items}})$ returns $Sol = \{\mathtt{A}^{\mathtt{C}}, \mathtt{B}^{\mathtt{C}}, \mathtt{C}, \mathtt{D}^{\mathtt{ABC}}, \mathtt{E}, \mathtt{AB}^{\mathtt{C}}\}$ where the notation $\mathtt{AB}^{\mathtt{C}}$ means that $\mathcal{C}_{Close}(\mathtt{AB})$ is true and that $\mathtt{closure}(\mathtt{AB}) = \mathtt{ABC}$ (this closure is returned by the Close algorithm together with AB and its frequency). We are then able to find all the frequent itemsets and their frequencies.

Proposition 6. *The* $\mathcal{C}_{Close}$ *constraint is anti-monotone.*

This constraint is another example of an anti-monotone constraint which needs a database pass to be checked (one needs a database pass to compute the closure of an itemset). However, checking this constraint seems expensive if the closure of every subset of S has to be computed. We can use an equivalent constraint $\mathcal{C}'_{Close}(S) \equiv (S' \subset S \wedge |S'| = |S| - 1) \Rightarrow S \not\subseteq \texttt{closure}(S')$. The equivalence means that $\mathcal{C}_{Close}(S)$ is true iff $\mathcal{C}'_{Close}(S)$ is true.

So we only need the closure of every subset of S of size $|S| - 1$. We are now able to test the constraint on $S \in \mathcal{L}_{k+1}$: for each $S' \subset S$ such that $|S'| = |S| - 1$ we must know $\texttt{closure}(S')$. In the original paper [14], the closure of each candidate itemset of size k and its frequency are computed during the database pass at level k. If the closure of S' was not computed, it would mean that S' does not verify $\mathcal{C}_{freq} \wedge \mathcal{C}_{Close}$ (i.e., an anti-monotone constraint) and therefore S cannot verify $\mathcal{C}_{freq} \wedge \mathcal{C}_{Close}$. Finally, either the closure of S' is known and we can check if $S \subseteq \texttt{closure}(S')$ or it is not known and it means that $\mathcal{C}_{freq}(S) \wedge \mathcal{C}_{Close}(S)$ is false. This strategy which uses the anti-monotonicity of $\mathcal{C}_{Close}$ enables to test the constraint with only a little extra cost during the database pass.

Now, it seems straightforward to search for itemsets which verify a constraint $\mathcal{C} = \mathcal{C}_{Close} \wedge \mathcal{C}'$ using the generic algorithm $\mathcal{G}$. If $\mathcal{C}'$ is anti-monotone, there is no problem: it is only a replacement of $\mathcal{C}_{freq}$ with another anti-monotone constraint (we have only used the anti-monotonicity of $\mathcal{C}_{freq}$ so far). Assume now the constraint is $\mathcal{C} = \mathcal{C}_{Close} \wedge \mathcal{C}_{am} \wedge \mathcal{C}_o$ where $\mathcal{C}_o$ is not anti-monotone. There are two problems. First, if we push $\mathcal{C}_o$, the closures of some candidates of level k will not be computed thus making the $\mathcal{C}_{Close}$ checking impossible at level $k+1$ when using the same strategy as in Close. Second, we loose the completeness of Close: $SAT_{\mathcal{C}_{Close} \wedge \mathcal{C}_{am} \wedge \mathcal{C}_o}$ will no longer enables to compute $SAT_{\mathcal{C}_{am} \wedge \mathcal{C}_o}$.

Assume we replace $\mathcal{C}_{Close}$ with $\mathcal{C}_{Close \wedge \mathcal{C}_o}(S) \equiv (S' \subset S \wedge \mathcal{C}_o(S')) \Rightarrow S \not\subseteq \texttt{closure}(S')$ and $\mathcal{C}'_{Close}$ with: $\mathcal{C}'_{Close \wedge \mathcal{C}_o}(S) \equiv (S' \subset S \wedge |S'| = |S| - 1 \wedge \mathcal{C}_o(S')) \Rightarrow S \not\subseteq \texttt{closure}(S')$

With these new constraints (and provided that they are equivalent which is false in the general case) the first problem is clearly solved: we only need the closure of itemsets that verify $\mathcal{C}_o$. The second one is also solved (the proof is omitted but is very similar to the completeness proof in [14]). However, $\mathcal{C}_{Close \wedge \mathcal{C}_o}$ is no longer anti-monotone in the general case and the anti-monotonicity of $\mathcal{C}_{Close}$ is the key point of the efficient strategy used to test $\mathcal{C}_{Close}$ and of the efficiency of the whole Close algorithm.

Fortunately, we can show that $\mathcal{C}_{Close \wedge \mathcal{C}_o}$ is equivalent to $\mathcal{C}'_{Close \wedge \mathcal{C}_o}$ and that they are anti-monotone if $\mathcal{C}_o$ is monotone.

Theorem 5. *If $\mathcal{C}_o$ is monotone then the constraints $\mathcal{C}_{Close \wedge \mathcal{C}_o}$ and $\mathcal{C}'_{Close \wedge \mathcal{C}_o}$ are equivalent and anti-monotone. The generic algorithm $\mathcal{G}$ with constraint $\mathcal{C} = \mathcal{C}_{am} \wedge \mathcal{C}_o \wedge \mathcal{C}_{Close \wedge \mathcal{C}_o}$ is complete in the sense that $SAT_{\mathcal{C}_{am} \wedge \mathcal{C}_o}$ can be computed from $SAT_{\mathcal{C}}$ and the closures of the itemsets in $SAT_{\mathcal{C}}$ (as in the original* Close *algorithm).*

It means that the Close algorithm can be extended to find frequent itemsets that verify conjunctions of anti-monotone and monotone constraints.

Example 11. Assume the frequency threshold $\gamma = 1/3$, the result of the query $\sigma_{\mathcal{C}_{Close} \wedge \mathcal{C}_{freq} \wedge \mathcal{C}}(2^{\mathtt{Items}})$ where $\mathcal{C}(S) \equiv \mathtt{A} \in S$ on our running example is $\{\mathtt{A}^{\mathtt{C}}, \mathtt{AB}^{\mathtt{C}}\}$.

6 Conclusion

We investigate the problem of optimizing the discovery of constrained itemsets. Unlike previous works that present how to push constraints during the search space exploration, we pointed out a tradeoff between pushing all the constraints and using a "generate and test" approach for some of the constraints.

Theorem 1 answers the problem for anti-monotone constraints and example 6 shows that a carefull study is necessary for non anti-monotone constraints. This is one obvious result of this paper and we gave some clues for the optimization of inductive queries. We focus on the case when the constraint is a conjunction of an anti-monotone constraint and a monotone one. Finally, we show how our framework can integrate the CLOSE algorithm and allow the discovery of constrained closed itemsets. Our framework is generic and can be used for other classes of patterns. For instance, we are currently working on its application to inclusion dependency discovery.

Furthermore, frequent itemsets discovery, and more generally data mining, is not limited to independent (inductive) queries. Knowledge discovery in databases is an iterative process and there are still a lot of work to do to optimize sequences of queries. There is a major tradeoff between fully optimizing each individual query and finding a strategy that makes use of previous mined patterns [7, 2]. This strategy may be less effective for the first queries but may win for long sequences of related queries, i.e., the way people actually proceed.

References

1. Rakesh Agrawal, Heikki Mannila, Ramakrishnan Srikant, Hannu Toivonen, and A. Inkeri Verkamo. Fast discovery of association rules. In Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, and Ramasamy Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 307 – 328. AAAI Press, Menlo Park, CA, 1996.
2. Elena Baralis and Giuseppe Psaila. Incremental refinement of mining queries. In Mukesh Mohania and A. Min Tjoa, editors, *Proceedings of the First International Conference on Data Warehousing and Knowledge Discovery (DaWaK'99)*, volume 1676 of *Lecture Notes in Computer Science*, pages 173 – 182, Florence, I, September 1999. Springer-Verlag.
3. Jean-François Boulicaut and Artur Bykowski. Frequent closures as a concise representation for binary data mining. In *Proceedings of the Fourth Pacif-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'00)*, volume 1805 of *Lecture Notes in Artificial Intelligence*, pages 62 – 73, Kyoto, JP, April 2000. Springer-Verlag.
4. Jean-François Boulicaut, Mika Klemettinen, and Heikki Mannila. Querying inductive databases: A case study on the MINE RULE operator. In Jan M. Zytkow and Mohamed Quafafou, editors, *Proceedings of the Second European Symposium on Principles of Data Mining and Knowledge Discovery (PKDD'98)*, volume 1510 of *Lecture Notes in Artificial Intelligence*, pages 194 – 202, Nantes, F, September 1998. Springer-Verlag.
5. Jean-François Boulicaut, Mika Klemettinen, and Heikki Mannila. Modeling KDD processes within the inductive database framework. In Mukesh Mohania and A. Min Tjoa, editors, *Proceedings of the First International Conference on Data Warehousing and Knowledge Discovery (DaWaK'99)*, volume 1676 of *Lecture Notes in Computer Science*, pages 293 – 302, Florence, I, September 1999. Springer-Verlag.

6. Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, and Ramasamy Uthurusamy, editors. *Advances in Knowledge Discovery and Data Mining*. AAAI Press, Menlo Park, CA, 1996.
7. Bart Goethals and Jan van den Bussche. On implementing interactive association rule mining. In *Proceedings of the ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD'99)*, Philadelphia, USA, May 30 1999.
8. Tomasz Imielinski and Heikki Mannila. A database perspective on knowledge discovery. *Communications of the ACM*, 39(11):58 – 64, November 1996.
9. Laks V.S. Lakshmanan, Raymond Ng, Jiawei Han, and Alex Pang. Optimization of constrained frequent set queries with 2-variable constraints. In *Proceedings of ACM SIGMOD Conference on Management of Data (SIGMOD'99)*, pages 157–168, Philadelphia, USA, 1999. ACM Press.
10. Heikki Mannila. Inductive databases and condensed representations for data mining. In Jan Maluszynski, editor, *Proceedings of the International Logic Programming Symposium (ILPS'97)*, pages 21 – 30, Port Jefferson, Long Island N.Y., USA, October 13-16 1997. MIT Press.
11. Heikki Mannila. Methods and problems in data mining. In *Proceedings of the International Conference on Database Theory (ICDT'97)*, volume 1186 of *Lecture Notes in Computer Science*, pages 41–55. Springer-Verlag, 1997.
12. Heikki Mannila and Hannu Toivonen. Levelwise search and borders of theories in knowledge discovery. *Data Mining and Knowledge Discovery*, 1(3):241 – 258, 1997.
13. Raymond Ng, Laks V.S. Lakshmanan, Jiawei Han, and Alex Pang. Exploratory mining and pruning optimization of constrained association rules. In *Proceedings of ACM SIGMOD Conference on Management of Data (SIGMOD'98)*, pages 13–24, Seattle, USA, 1998. ACM Press.
14. Nicolas Pasquier, Yves Bastide, Rafik Taouil, and Lotfi Lakhal. Efficient mining of association rules using closed itemset lattices. *Information Systems*, 24(1):25 – 46, January 1999.
15. Ramakrishnan Srikant, Quoc Vu, and Rakesh Agrawal. Mining association rules with item constraints. In David Heckerman, Heikki Mannila, Daryl Pregibon, and Ramasamy Uthurusamy, editors, *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining (KDD'97)*, pages 67 – 73. AAAI Press, 1997.