

Interactive Constrained Association Rule Mining*

Bart Goethals[†]

Helsinki Institute for Information Technology

Jan Van den Bussche

University of Limburg

Abstract

We investigate ways to support interactive mining sessions, in the setting of association rule mining. In such sessions, users specify conditions (queries) on the associations to be generated. Our approach is a combination of the integration of querying conditions inside the mining phase, and the incremental querying of already generated associations. We present several concrete algorithms and compare their performance.

1 Introduction

The interactive nature of the mining process has been acknowledged from the start [5]. It motivated the idea of a “data mining query language” [8, 9, 12, 13, 19] and was stressed again by Ng, Lakshmanan, Han and Pang [21]. A data mining query language allows the user to ask for specific subsets of association rules by specifying several constraints within each query.

In this paper, working in the concrete setting of association rule mining, we consider a class of conditions on associations to be generated which should be expressible in any reasonable data mining query language: Boolean combinations of atomic conditions, where an atomic condition can either specify that a certain item occurs in the body of the rule or the head of the rule, or set a threshold on the support or on the confidence. A *mining session* then consists of a sequence of such Boolean combinations (henceforth referred to as *queries*). Efficiently supporting data mining query language environments is a challenging task. Towards this goal, we present and compare three approaches. In the first extreme, the *integrated querying* approach, every individual data mining query will be answered by running an adaptation of the mining algorithm in which the constraints on the rules and sets to be generated are directly incorporated.

*A preliminary report on this work was presented at the Second International Conference on Knowledge Discovery and Data Mining [16].

[†]This work was done while the author was employed by the University of Limburg

The second extreme, the *post-processing* approach, first mines as much associations as possible, by performing one major, global mining operation. After this relatively expensive operation, the actual data mining queries issued by the user then amount to standard lookups in the set of materialized associations. A third approach, the *incremental querying* approach, combines the advantages of both previous approaches.

1.1 Our contributions

We present the first algorithm to support interactive mining sessions efficiently. We measure efficiency in terms of the total number of itemsets that are generated, but do not satisfy the query, and the number of scans over the database that have to be performed. Specifically, our results are the following:

1. Although our results show significant improvements of performance, we will also show that exploiting constraints is not always the best solution. More specifically, if mining without constraints is feasible to begin with, then the presented post-processing approach will eventually outperform integrated querying.
2. The querying achieved by exploiting the constraints is *optimal*, in the sense that it never generates an itemset that could give rise to a rule that does not satisfy the query, apart from the minimal support and confidence thresholds. Therefore, the number of generated itemsets during the execution of a query, becomes proportional to the strength of the constraints in the query: the more specific the query, the faster its execution.
3. Not only is the number of passes through the database reduced, but also the size of the database itself, again proportionally to the strength of the constraints in the query.
4. A generated itemset will, within a session, never be regenerated as a candidate itemset: results of earlier queries are reused when answering a new query.

This paper is further organized as follows. Section 2 gives an overview of related work on constrained mining. In Section 4, we present a way of incorporating query-constraints inside a frequent set mining algorithm. In Section 5, we discuss ways of supporting interactive mining sessions. We conclude the paper in Section 6.

2 Related Work

The idea that queries can be integrated in the mining algorithm was initially launched by Srikant, Vu, and Agrawal [25], who considered queries that are Boolean expressions over the presence or absence of certain items in the rules.

Queries specifically as bodies or heads were not discussed. The authors considered three different approaches to the problem. The proposed algorithms are not optimal: they generate and test several itemsets that do not satisfy the query, and their optimizations also do not always become more efficient for more specific queries.

Also Lakshmanan, Ng, Han and Pang worked on the integration of constraints on itemsets in mining, considering conjunctions of conditions on itemsets such as those considered here, as well as others (arbitrary Boolean combinations were not discussed) [18, 21]. Of the various strategies for the so-called “CAP” algorithm they present, the one that can handle the queries considered in the present paper is their “strategy II”. Again, this strategy generates and tests itemsets that do not satisfy the query. Also, their algorithms implement a rule-query by separately mining for possible heads and for possible bodies, while we tightly couple the querying of rules with the querying of sets. This work has also been further studied by Pei, Han and Lakshmanan [22, 23], and employed within the FPgrowth algorithm.

Still other work focused on other kinds of constraints over association rules and frequent sets, such as *correlation* [7], and *improvement* [15]. These and other statistical measures of interestingness will not be discussed in this paper.

All previously mentioned works do not discuss the reuse of results acquired from earlier queries within a session. Nag et al. proposed the use of a knowledge cache for this purpose [20]. Several caching strategies were studied for different cache sizes. However, their work only considers mining sessions of queries where only constraints on the support of the itemsets are allowed. No solutions were provided for other constraints like those studied in this paper. Also Jeudy and Boulicaut have studied the use of a knowledge cache for finding a condensed representation of all itemsets, based on the concept of *free sets* [14].

3 Review of the Apriori algorithm

As introduced by Agrawal et al. [1], the association rule mining problem can be described as follows: we are given a set of items \mathcal{I} and a database \mathcal{D} of subsets of \mathcal{I} called transactions. An association rule is an expression of the form $B \Rightarrow H$, where B and H are sets of items (itemsets). The *support* of an itemset I is the number of transactions that include I . An itemset is called *frequent* if its support is no less than a given minimal support threshold. An association rule is called frequent if $B \cup H$ is frequent and it is called *confident* if the support of $B \cup H$ divided by the support of B exceeds a given minimal confidence threshold. The goal is now to find all association rules over \mathcal{D} that are frequent and confident.

The standard association rule mining algorithm Apriori [2] is divided in two phases: phase 1 generates all frequent itemsets with respect to the given minimal support threshold, and phase 2 generates all confident rules with respect to the given minimal confidence threshold.

Phase 1 is performed based on the observation (also called the anti-monotonicity

property) that all supersets of an infrequent itemset are also infrequent. An itemset is thus potentially frequent, also called a *candidate* itemset, if its support is unknown and all of its subsets are frequent. In every step of the algorithm, all candidate itemsets are generated and their supports are then counted by performing a complete scan of the transaction database. This is repeated until no new candidate itemsets can be generated.

Phase 2 generates for every frequent itemset a set of rules by dividing the itemset in potential bodies and heads. This can be done in a similar level-wise manner as in phase 1, based on the observation that if a head-set represents a confident rule for that itemset, then all of its subsets also represent confident rules [24]. For example, if the itemset $\{1, 2, 3, 4\}$ is a frequent set and $\{1, 2\} \Rightarrow \{3, 4\}$ is a confident rule, then $\{1, 2, 3\} \Rightarrow \{4\}$ and $\{1, 2, 4\} \Rightarrow \{3\}$ must also be confident. In every step within phase 2, all candidate head-sets are generated and their confidences are computed, until no new candidate head-sets can be generated. Because we do not need to access the database, phase 2 is much faster in comparison with phase 1.

The performance of Apriori-like algorithms is highly dependent of three factors:

1. the number of candidate patterns increases exponentially with a decreasing minimal support threshold,
2. the number of association rules can become very large for small confidence thresholds, and
3. the size of the transaction database is typically very large, such that scanning the database becomes a costly operation.

Also the length of the transactions (density of the database) plays an important role, because large transactions can result in large frequent patterns, implying a lot of candidate patterns and a lot of scans through the database. Since the introduction of the Apriori algorithm, a lot of research has been done to improve its performance by improving on one or more of these factors. Almost all improvements rely on its levelwise, bottom-up, breadth-first nature and on the anti-monotonicity property of the minimal support threshold.

Nevertheless, Han et al. presented the FPgrowth algorithm [10], which uses a depth-first strategy. Although this algorithm has a very efficient counting mechanism, it suffers from two major deficiencies:

1. it cannot exploit the anti-monotonicity property, resulting in a lot more candidate patterns, and
2. although the used trie data structure somewhat compresses the transaction database, the algorithm implicitly requires the database to reside in main-memory.

Although recent improvements have increased the performance of Apriori tremendously, the support and confidence thresholds can always be set low

enough, resulting in an exponential blowup of the number of patterns and rules. Nevertheless, such low thresholds can still reveal interesting patterns and rules, but one is not interested in all of the discovered patterns and rules, but queries out the interesting ones according to some specified constraints. Pushing these constraints as deep as possible into the mining algorithm, such that the amount of computation is proportional to what the user gets, should improve its performance and allow lower thresholds.

4 Exploiting Constraints

As already mentioned in the Introduction, the constraints we consider in this paper are Boolean combinations of atomic conditions. An atomic condition can either specify that a certain item i occurs in the body of the rule or the head of the rule, denoted respectively by $\text{Body}(i)$ or $\text{Head}(i)$, or set a threshold on the support or on the confidence.

In this section, we explain how we can incorporate these constraints in the mining algorithm. We first consider the special case of constraints where only conjunctions of atomic conditions or their negations are allowed.

4.1 Conjunctive Constraints

Let b_1, \dots, b_ℓ be the items that must be in the body by the constraint; $b'_1, \dots, b'_{\ell'}$ those that must not; h_1, \dots, h_m those that must be in the head; and $h'_1, \dots, h'_{m'}$ those that must not.

Recall that an association rule $X \Rightarrow Y$ is only generated if $X \cup Y$ is a frequent set. Hence, we only have to generate those frequent sets that contain every b_i and h_i , plus some of the subsets of these frequent sets that can serve as bodies or heads. Therefore we will create a set-query corresponding to the rule-query, which is also a conjunctive expression, but now over the presence or absence of an item i in a frequent set, denoted by $\text{Set}(i)$ and $\neg\text{Set}(i)$. We do this as follows:

1. For each positive literal $\text{Body}(i)$ or $\text{Head}(i)$ in the rule-query, add the literal $\text{Set}(i)$ in the set-query.
2. If for an item i both $\neg\text{Body}(i)$ and $\neg\text{Head}(i)$ are in the rule-query, add the negated literal $\neg\text{Set}(i)$ to the set-query.
3. Add the minimal support threshold to the set-query.
4. All other literals in the rule-query are ignored because they do not restrict the frequent sets that must be generated.

Formally, the following is readily verified:

Lemma 1. *An itemset Z satisfies the set-query if and only if there exists itemsets X and Y such that $X \cup Y = Z$ and the rule $X \Rightarrow Y$ satisfies the rule-query, apart from the confidence threshold.* \square

So, once we have generated all sets Z satisfying the set-query, we can generate all rules satisfying the rule-query by splitting all these Z in all possible ways in a body X and a head Y such that the rule-query is satisfied. Lemma 1 guarantees that this method is “sound and complete”.

So, we need to explain two things:

1. Finding all frequent Z satisfying the set-query.
2. Finding, for each such Z , the frequencies of all bodies and heads X and Y such that $X \cup Y = Z$ and $X \Rightarrow Y$ satisfies the rule-query.

Finding the frequent sets satisfying the set-query Let $Pos := \{i \mid \text{Set}(i) \text{ in set-query}\}$ and $Neg := \{i \mid \neg \text{Set}(i) \text{ in set-query}\}$. Note that $Pos = \{b_1, \dots, b_\ell, h_1, \dots, h_m\}$. Denote the dataset of transactions by \mathcal{D} . We define the following derived dataset \mathcal{D}_0 :

$$\mathcal{D}_0 := \{t - (Pos \cup Neg) \mid t \in \mathcal{D} \text{ and } Pos \subseteq t\}$$

In other words, we ignore all transactions that are not supersets of Pos and from all transactions that are not ignored, we remove all items in Pos plus all items that are in Neg .

We observe:

Lemma 2. *Let p be the support threshold defined in the query. Let \mathcal{S}_0 be the set of itemsets over the new dataset \mathcal{D}_0 , without any further conditions, except that their support is at least p . Let \mathcal{S} be the set of itemsets over the original dataset \mathcal{D} that satisfy the set-query, and whose support is also at least p . Then*

$$\mathcal{S} = \{s \cup Pos \mid s \in \mathcal{S}_0\}.$$

Proof. To show the inclusion from left to right, consider $Z \in \mathcal{S}$. We show that $s := Z - Pos$ is in \mathcal{S}_0 . Thereto, it suffices to establish an injection $t \mapsto t_0$ from the transactions t in the support set of Z in \mathcal{D} (i.e., the set of all transactions in \mathcal{D} containing Z) into the transactions t_0 in the support set of s in \mathcal{D}_0 .

Let t be in \mathcal{D} and containing Z . Since Z satisfies the set-query, Z contains Pos , and hence t contains Pos as well. Thus, $t_0 := t - (Pos \cup Neg)$ is in \mathcal{D}_0 . Since $Z \cap Neg = \emptyset$ (again because Z satisfies the set-query), t_0 contains $Z - Pos = s$. Hence, t_0 is in the support set of s in \mathcal{D}_0 , as desired.

To show the inclusion from right to left, consider $s \in \mathcal{S}_0$. We show that $Z := s \cup Pos$ is in \mathcal{S} . Thereto, it suffices to establish an injection $t_0 \mapsto t$ from the transactions t_0 in the support set of s in \mathcal{D}_0 into the transactions t in the support set of Z in \mathcal{D} .

Let t_0 be in \mathcal{D}_0 and containing s . Obviously, a transaction $t \in \mathcal{D}$ exists, such that $t_0 \cup Pos \subseteq t - Neg \subseteq t$. Since t contains $s \cup Pos$, it is in the support set of Z in \mathcal{D} , as desired. \square

We can thus perform any frequent set generation algorithm, using only \mathcal{D}_0 instead of \mathcal{D} . Note that the number of transactions in \mathcal{D}_0 is exactly the support

of Pos in \mathcal{D} . Also, the search space of all itemsets is halved for every item in $Pos \cup Neg$. In practice, the search space of all frequent itemsets is at least halved for every item in Pos and at most halved for every item in Neg . Still put differently: we are mining in a world where itemsets that do not satisfy the query simply do not exist. The correctness and optimality of our method is thus automatically guaranteed.

Note however that now an itemset I , actually represents the itemset $I \cup Pos$! We thus head-start with a lead of k , where k is the cardinality of Pos , in comparison with standard, non-constrained mining.

Finding the frequencies of bodies and heads We now have all frequent sets containing every b_i and h_i , from which rules that satisfy the rule-query can be generated. Recall that in the standard association rule mining algorithm rules are generated by taking every item in a frequent set as a head and the others as body. All heads that result in a confident rule, with respect to the minimal confidence threshold, can then be combined to generate more general rules. But, because we now only want rules that satisfy the query, a head must always be a superset of $\{h_1, \dots, h_m\}$ and may not include any of the h'_i and b_i (the latter because bodies and heads of rules are disjoint). In this way, we head-start with a lead of m . Similarly, a body must always be a superset of $\{b_1, \dots, b_\ell\}$ and may not include any of the b'_i and h_i .

The following lemma (which follows immediately from Lemma 2) tells us that these potential heads and bodies are already present, albeit implicitly, in \mathcal{S}_0 :

Lemma 3. *Let \mathcal{S}_0 be as in Lemma 2. Let $\mathcal{B}(\mathcal{H})$ be the set of bodies (heads) of those association rules over \mathcal{D} that satisfy the rule-query. Then*

$$\mathcal{B} = \{s \cup \{b_1, \dots, b_\ell\} \mid s \in \mathcal{S}_0 \text{ and } s \cap \{b'_1, \dots, b'_\ell, h_1, \dots, h_m\} = \emptyset\}$$

and

$$\mathcal{H} = \{s \cup \{h_1, \dots, h_m\} \mid s \in \mathcal{S}_0 \text{ and } s \cap \{h'_1, \dots, h'_m, b_1, \dots, b_\ell\} = \emptyset\}.$$

So, for the potential bodies (heads), we use, in \mathcal{S}_0 , all sets that do not include any of the b'_i and h_i (h'_i and b_i), and add all b_i (h_i). Hence, all we have to do is to determine the frequencies of these subsets by performing one additional scan through the dataset. (We do not necessarily yet have these frequencies because these sets do not contain either items b_i or h_i , while we ignored transactions that did not contain all items b_i and h_i .)

Each generated itemset can thus have up to three different “personalities:”

1. A frequent set that satisfies the set-query;
2. A frequent set that can act as body of a rule that satisfies the rule-query;
3. A frequent set that can act as head of a rule that satisfies the rule-query.

Hence, we finally have at most three families of sets, i.e., those sets from which rules must be generated, the *rule-sets* (\mathcal{S}_0 with all b_i and h_i added); a family of possible bodies, the *body-sets* (\mathcal{S}_0 with all b_i added, minus all those sets that include any of the b'_i and h_i); and yet another family of possible heads, the *head-sets* (\mathcal{S}_0 with all h_i added, minus all those sets that include any of the h'_i and b_i). Note that the frequencies of the body-sets and head-sets need not necessarily to be recounted since their frequencies are equal to the frequencies of their corresponding sets in \mathcal{S}_0 if the query consists of negated atoms only. We finally generate the desired association rules from the rule-sets, by looking for possible bodies and heads only within the body-sets and head-sets respectively, on condition that they have enough confidence.

Optimality Note that every rule-set, body-set, and head-set is needed to construct the rules potentially satisfying the rule-query so that these can be tested for confidence, and moreover, no other sets are ever needed. In this precise sense, our method is *optimal*.

Example 1. *We illustrate our method with an example. Assume we are given the rule-query*

$$\begin{aligned} &\text{Body}(1) \wedge \neg \text{Body}(2) \wedge \text{Head}(3) \wedge \neg \text{Head}(4) \\ &\quad \wedge \neg \text{Body}(5) \wedge \neg \text{Head}(5) \wedge \text{support} \geq 1 \wedge \text{confidence} \geq 50\%. \end{aligned}$$

We begin by converting it to the set-query

$$\text{Set}(1) \wedge \text{Set}(3) \wedge \neg \text{Set}(5) \wedge \text{support} \geq 1.$$

Hence $\text{Pos} = \{1, 3\}$ and $\text{Neg} = \{5\}$. Consider a database consisting of the three transactions $\{2, 3, 5, 6, 9\}$, $\{1, 2, 3, 5, 6\}$ and $\{1, 3, 4, 8\}$. We ignore the first transaction because it is not a superset of Pos . We remove items 1 and 3 from the second transaction because they are in Pos , and we also remove 5 because it is in Neg . We only remove items 1 and 3 from the third transaction. Table 1 shows the itemsets that result from the mining algorithm after reading, according to Lemma 1 and 2, the two resulting transactions. For example, the itemset $\{4, 8\}$ actually represents the set $\{1, 3, 4, 8\}$. It also represents a potential body, namely $\{1, 4, 8\}$, but it does not represent a head, because it includes item 4, which must not be in the head according to the given rule-query. As another example, the empty set now represents the set $\{1, 3\}$ from which a rule can be generated. It also represents a potential body and a potential head.

4.2 Boolean Queries

Assume now given a rule-query that is an arbitrary Boolean combination of atomic conditions. We can put it in disjunctive normal form and then generate all frequent itemsets for every disjunct (which is a conjunction) in parallel by

\mathcal{S}_0	\mathcal{S}	\mathcal{B}	\mathcal{H}
$\{\}$	$\{1, 3\}$	$\{1\}$	$\{3\}$
$\{2\}$	$\{1, 2, 3\}$	-	$\{2, 3\}$
$\{4\}$	$\{1, 3, 4\}$	$\{1, 4\}$	-
$\{6\}$	$\{1, 3, 6\}$	$\{1, 6\}$	$\{3, 6\}$
$\{8\}$	$\{1, 3, 8\}$	$\{1, 8\}$	$\{3, 8\}$
$\{2, 6\}$	$\{1, 2, 3, 6\}$	-	$\{2, 3, 6\}$
$\{4, 8\}$	$\{1, 3, 4, 8\}$	$\{1, 4, 8\}$	-

Table 1: An example of generated sets, which can represent a frequent set, as well as a body, as well as a head

feeding every transaction of the database to every disjunct, and processing them there as described in the previous subsection.

However, this approach is a bit simplistic, as it might generate some sets and rules multiple times. For example, consider the following query: $\text{Body}(1) \vee \text{Body}(2)$. If we convert it to its corresponding set-query (disjunct by disjunct), we get $\text{Set}(1) \vee \text{Set}(2)$. Then, we would generate for both disjuncts all supersets of $\{1, 2\}$. We can avoid this problem by putting the set-query to *disjoint DNF*.¹ Then, no itemset can satisfy more than one set-disjunct. On the other hand this does not solve the problem of generating some rules multiple times. Consider the equivalent disjoint DNF of the above set-query: $\text{Set}(1) \vee (\text{Set}(2) \wedge \neg \text{Set}(1))$. The first disjunct thus contains the set $\{1, 2\}$ and all of its supersets. If we generate for every itemset all potential bodies and heads according to every rule-disjunct, both rule-disjuncts will still generate all rules with the itemset $\{1, 2\}$ in the body. The easiest way to avoid this problem is to put already the rule-query in disjoint DNF. Obviously, this does not mean its corresponding set-query is also in disjoint DNF, and hence, we still have to put it in disjoint DNF.

After all sets have been generated according to the set-query, we still have to generate all rules according to the rule-query. This can be done for every rule-disjunct (which is a conjunction) in parallel after some modifications to the algorithm described in the previous subsection.

Indeed, a single set-disjunct can now contain sets from which rules can be generated satisfying several rule-disjuncts. Hence, a set generated in one set-disjunct has now possibly even more personalities. More specifically, it can possibly represent for every rule disjunct a set from which rules can be generated, a body of a such a rule and a head of such a rule. We illustrate this with the rule-query given in the previous paragraph.

Example 2. Assume we are given the rule-query

$$\text{Body}(1) \wedge (\text{Body}(2) \vee \text{Head}(2)).$$

¹In disjoint DNF, the conjunction of any two disjuncts is unsatisfiable. Any boolean expression has an equivalent disjoint DNF.

In disjoint DNF, this gives

$$(\text{Body}(1) \wedge \text{Body}(2)) \vee (\text{Body}(1) \wedge \text{Head}(2)).$$

Converted to its corresponding set-query in disjoint DNF, we get

$$\text{Set}(1) \wedge \text{Set}(2).$$

Obviously, this single set-disjunct contains sets from which rules satisfying the first rule-disjunct can be generated. Following the methodology described in the previous subsection, this means we still have to count the frequencies of all these sets without item 1 and item 2 included, since they will occur as heads in the rules satisfying the first rule disjunct. But now, the set-disjunct also contains sets from which rules satisfying the second rule-disjunct can be generated. Hence, we still have to count the frequencies of the generated sets with item 1 included, which can serve as bodies for the rules satisfying the second rule-disjunct, and the sets with item 2 included, which can serve as heads.

Until now, we have disregarded the possible presence of negated thresholds in the queries, which can come from the conversion to disjoint DNF, or from the user himself. In the latter case, it would not be possible to exploit this constraint in an Apriori-like algorithm, because it is an essentially bottom-up algorithm. Algorithms that generate sets also in a top-down strategy could exploit this constraint. Another source for negated thresholds is the conversion from the user's query to a Disjoint DNF formula. Before we discuss this, we first have to explain how we are going to convert a given formula to disjoint DNF.

We first put the Boolean expression ϕ in DNF, obtaining an expression of the form $\phi_1 \vee \phi_2 \vee \dots \vee \phi_n$, in which ϕ_i is a conjunction of atomic conditions or their negations. Of course, any two of these disjuncts may not be disjoint. A good way to obtain a disjoint DNF is to add to every disjunct ϕ_i the negated disjuncts ϕ_j with $j < i$. We thus become the equivalent formula $\phi_1 \vee (\phi_2 \wedge \neg\phi_1) \vee \dots \vee (\phi_n \wedge \neg\phi_{n-1} \wedge \dots \wedge \neg\phi_1)$ in which all disjuncts are pairwise disjoint. Our problem is not yet solved, because our formula is not even in DNF anymore. We thus still will have to convert every disjunct on itself to disjoint DNF. For example, take $(\phi_2 \wedge \neg\phi_1)$ with $\phi_1 \equiv p_1 \wedge p_2 \wedge \dots \wedge p_\ell$ in which p_i is an atomic condition or its negation. The disjunct thus becomes $(\phi_2 \wedge \neg p_1) \vee (\phi_2 \wedge p_1 \wedge \neg p_2) \vee \dots \vee (\phi_2 \wedge p_1 \wedge p_2 \wedge \dots \wedge p_{\ell-1} \wedge \neg p_\ell)$, which is in disjoint DNF.

An example showing that negated thresholds can be introduced in this process, is the following.

Example 3. Assume we are given the rule-query

$$(\text{Body}(1) \wedge \text{support} \geq 10) \vee (\text{Body}(2) \wedge \text{support} \geq 5).$$

As equivalent disjoint DNF, we obtain

$$\begin{aligned} &(\text{Body}(1) \wedge \text{support} \geq 10) \vee (\text{Body}(2) \wedge \text{support} \geq 5 \wedge \neg\text{Body}(1)) \\ &\vee (\text{Body}(2) \wedge \text{support} \geq 5 \wedge \text{Body}(1) \wedge \text{support} < 10). \end{aligned}$$

Data set	#Items	#Transactions	MinSup	It's	Time
T40I10D100K	1 000	100 000	700	18	1 700s
mushroom	120	8 124	813	16	663s
BMS-Webview-1	498	59 602	36	15	86s
basket	13 103	41 373	5	11	43s

Table 2: Data set Characteristics

Notice the maximal support threshold in the last disjunct, which is needed to avoid generating itemsets satisfying $\text{Body}(2) \wedge \text{support} \geq 10 \wedge \text{Body}(1)$ which are already generated by the first disjunct.

Negated support thresholds can be avoided however. After putting the user's formula in DNF, but before putting the DNF in disjoint DNF, we sort all disjuncts on their support threshold, in ascending order. This guarantees that the conversion to disjoint DNF does not introduce any negated support thresholds.

Note that we cannot avoid negated *confidence* thresholds at the same time: we have already sorted on support, and thus cannot sort anymore on confidence at the same time. Since we are here already in phase 2, it is less of an efficiency issue to just ignore maximal confidence thresholds.

Furthermore, if a set-disjunct (rule-disjunct) consists of nothing but a negated support (confidence) threshold, we can of course easily switch the generation algorithm and generate the candidate sets (heads) in a top-down manner.

4.3 Experiments

For our experiments, we have implemented an extensively optimized version of the Apriori algorithm, equipped with the querying optimizations as described in the previous sections.

We have experimented using three real data sets, of which two are publicly available and one synthetic data set generated by the program provided by the Quest research group at IBM Almaden [3]. The mushroom data set contains characteristics of various species of mushrooms, and was originally obtained from the UCI repository of machine learning databases [4]. The BMS-WebView-1 data set contains several months worth of click-stream data from an e-commerce web site, and is made publicly available by Blue Martini Software [17]. The basket data set contains transactions from a Belgian retail store, but can unfortunately not be made publicly available. Table 2 shows the number of items and the number of transactions in each data set. The table additionally shows the minimal support threshold we used in our experiments for each data set, together with the resulting number of iterations and the time (in seconds) which the Apriori algorithm needed to find all frequent patterns.

For each data set, we generated 100 random Boolean queries consisting of at most three atomic conditions. Figure 1 shows the improvement on the performance of the algorithm exploiting the constraints. The y-axis shows the time needed for the algorithm exploiting our queries, relative to the time needed with-

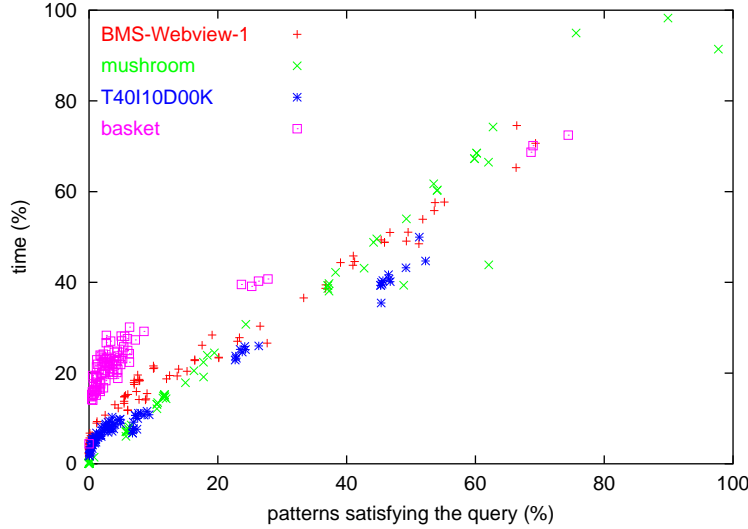


Figure 1: Improvement after exploiting constraints.

out exploiting the queries. The x-axis shows the number of patterns satisfying the given query, relative to the total number of patterns. As can be seen, the time needed to generate all frequent sets and association rules is proportional to the restrictiveness of the constraints. Notice that the proportionality factor is 1.

5 Interactive Mining

5.1 Integrated Querying or Post-Processing?

In the previous section, we have seen a way to integrate constraints tightly into the mining of association rules. We call this *integrated querying*. At the other end of the spectrum we have *post-processing*, where we perform standard, non-constrained mining, save the resulting itemsets and rules, and then query those results for the constraints.

Integrated querying has the following two obvious advantages over post-processing:

1. Answering one single data mining query using integrated querying is much more efficient than answering it using post-processing.
2. It is well known that, by setting parameters such as minimal support too low, or by the nature of the data, association rule mining can be infeasible simply because of a combinatorial explosion involved in the generation of rules or frequent itemsets. Under such circumstances, of course, post-processing is infeasible as well; yet, integrated querying can still be

executed, if the query conditions can be effectively exploited to reduce the number of itemsets and rules from the outset.

However, as already mentioned in the Introduction, data mining query language environments must support an interactive, iterative mining process, where a user repeatedly issues new queries based on what he found in the answers of his previous queries. Now consider a situation where minimal support requirements and data set particulars are favorable enough so that post-processing is not infeasible to begin with. Then the global, non-constrained mining operation, on the result of which the querying will be performed by post-processing, *can be executed once and its result materialized for the remainder of the data mining session.*

In that case, if the session consists of, say, 20 data mining queries, these 20 queries amount to standard retrieval queries on the materialized mining results. In contrast, answering every single of the 20 queries by integrated querying will involve at least 20, and often many more, passes over the data, as each query involves a separate mining operation. Also, several queries could have a non-empty intersection, such that a lot of work is repeated several times. Hence, the total time needed to answer the integrated queries is guaranteed to grow beyond the post-processing total time.

The naively conceived advantages of integrated querying over post-processing become much less clear now. Indeed, if the number of data mining queries issued by the user is large enough, then the post-processing approach clearly outperforms the integrated querying approach. We have performed several experiments on the data sets described in the previous section which all confirmed this predicted effect. However, for the post-processing approach, we only materialized all frequent itemsets since the time needed to generate all association rules that satisfy the query turned out to be as fast as finding all such rules from the materialized results. Figure 2 shows the total time needed for answering up to 20 different queries on the BMS-Webview-1 data set. Since the time needed to generate all association rules is the same for both approaches, we only recorded the time to generate all itemsets that were needed to generate all association rules. The queries were randomly generated, only those queries with an empty output were replaced, but all used the same support threshold as was used for the initial mining operation of the post-processing approach. As can be seen, the cut-off point from where the post-processing approach outperforms the integrated querying approach occurs already after the eighth query.

5.2 Incremental Querying: Basic Approach

From the above discussion it is clear that we should try to combine the advantages of integrated querying and post-processing. We now introduce such an approach, which we call *incremental querying*.

In the incremental approach, all itemsets that result from every posed query, as well as all intermediate generated itemsets, are stored into a cache. Initially,

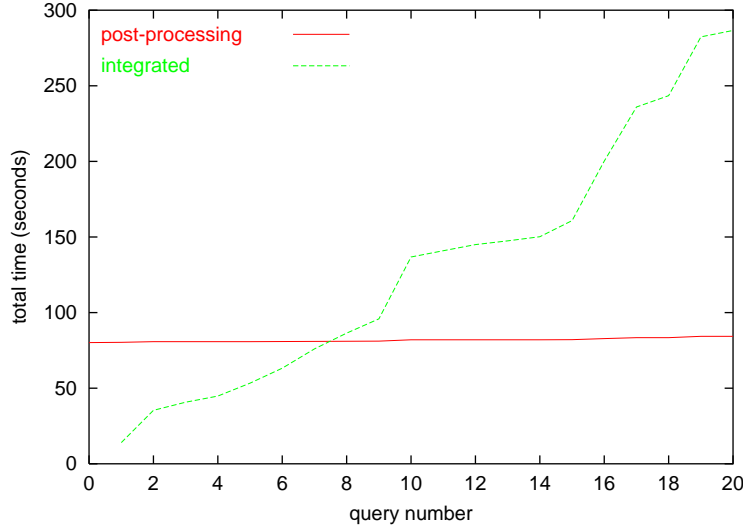


Figure 2: Integrated querying versus post-processing.

when the user issues his first query, nothing has been mined yet, and thus we answer it using integrated querying.

Every subsequent query is first converted to its corresponding rule- and set-query in disjoint DNF. For every disjunct in the set-query, the system adds all currently cached itemsets that satisfy the disjunct to the data structure holding itemsets, that is used for mining that disjunct, as well as all of its subsets that satisfy the disjunct (note that these subsets may not all be cached; if they are not, we have to count their supports during the first scan through the data set). We also immediately add all candidate itemsets.

If no new candidate itemsets can be generated, which means that all necessary itemsets were already cached, we are done. However, if this is not the case, we can now begin our queried mining algorithm with the important generalization that in each iteration, candidate itemsets of different cardinalities are now generated. In order for this to work, candidate itemsets that turn out to be infrequent must be kept such that they are not regenerated in later iterations. This generalization was first used by Toivonen in his sampling algorithm [26].

Caching all generated itemsets gives us another advantage that can be exploited by the integrated querying algorithm. Consider a set-query stating that items 1 and 2 must be in the itemsets. In the first iteration of the algorithm, all single itemsets are generated as candidate sets over the new data set \mathcal{D}_0 (cf. Section 4.1). We explained that these single itemsets actually represent supersets of $\{1, 2\}$. Normally, before we generate a candidate itemset, we check if all of its subsets are frequent. Of course, this is impossible if these subsets do not even exist in \mathcal{D}_0 . Now, however, we can check in the cache for a subset with too low support; if we find this, we avoid generating the candidate.

We thus obtain an algorithm which reuses previously generated itemsets as if they had been generated in previous iterations of the algorithm. We are optimal in the sense that we never generate and test itemsets that were generated before. For rule generation, we again did not cache the results, but instead generated all association rules when needed for the same reasons as explained in the previous section.

In the worst case, the cached results do not contain anything that can be reused for answering a query, and hence the time needed to generate the itemsets and rules that satisfy the query is equal to the time needed when answering that query using the integrated querying approach. In the best case, all requested itemsets are already cached, and hence the time needed to find all itemsets and rules that satisfy the query is equal to the time needed for answering that query using post-processing. In the average case, part of the needed itemsets are cached and will then be used to speed up the integrated querying approach. If the time gained by this speedup is more than the time needed to find the reusable sets, then the incremental approach will always be faster than the integrated querying approach. In the limit, all itemsets will be materialized, and hence all subsequent queries will be answered using post-processing.

5.3 Incremental Querying: Overhead

Could it be that the time gained by the speedup in the integrated querying approach is less than the time needed to find and reuse the reusable itemsets? This could happen when a lot of itemsets are already cached, but almost none of them satisfy the constraints. It is also possible that the reusable itemsets give only a marginal improvement. We can however counter this phenomenon by estimating what is currently cached, as follows.

We keep track of a set-query ϕ_{sets} which describes the stored sets. This query is initially *false*. Given a new query (rule-query) ψ , the system now goes through the following steps: (step 1 was described in Section 4.1)

1. Convert the rule-query ψ to the set-query ϕ
2. $\phi_{mine} := \phi \wedge \neg\phi_{sets}$
3. $\phi_{sets} := \phi_{sets} \vee \phi$

After this, we perform:

1. Generate all frequent sets according to ϕ_{mine} , using the basic incremental approach.
2. Retrieve all cached sets satisfying $\phi \wedge \neg\phi_{mine}$.
3. Add all needed subsets that can serve as bodies or heads.
4. Generate all rules satisfying ψ .

Note that the query ϕ_{mine} is much more specific than the original query ϕ . We thus obtain a speedup, because we have shown in Section 4 that the speed of integrated querying is proportional to the restrictiveness of the query.

5.4 Avoiding Exploding Queries

The improvement just described incurs a new problem. The formula ϕ_{sets} becomes longer with the session. When, given the next query ϕ , we mine for $\phi \wedge \neg\phi_{sets}$, and convert this to disjoint DNF which could explode.

To avoid this, consider ϕ_{sets} in DNF: $\phi_1 \vee \dots \vee \phi_n$. Instead of the full query $\phi \wedge \neg\phi_{sets}$, we are going to use a query $\phi \wedge \neg\phi'_{sets}$, where ϕ'_{sets} is obtained from ϕ_{sets} by keeping only the least restrictive disjuncts ϕ_i (their negation will thus be most restrictive). In this way $\phi \wedge \neg\phi'_{sets}$ is kept short.

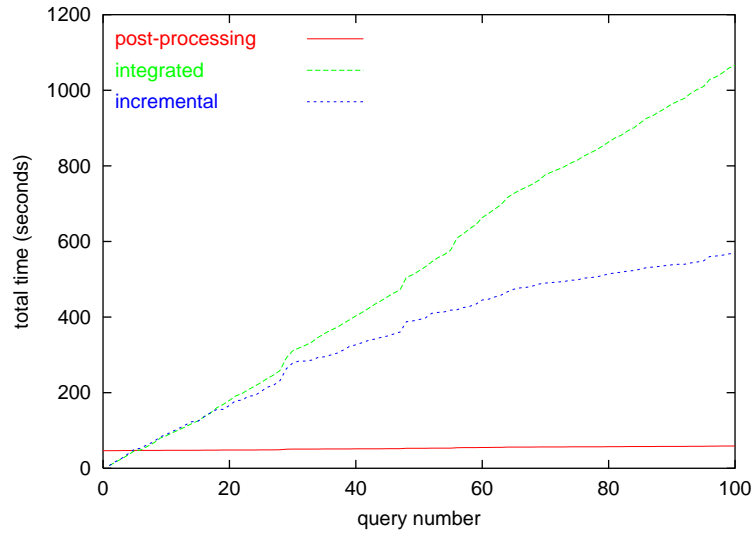
But how do we measure restrictiveness of a ϕ_i ? Several heuristics come to mind. A simple one is to keep for each ϕ_i the number of cached sets that satisfy it. These numbers can be maintained incrementally.

5.5 Experiments

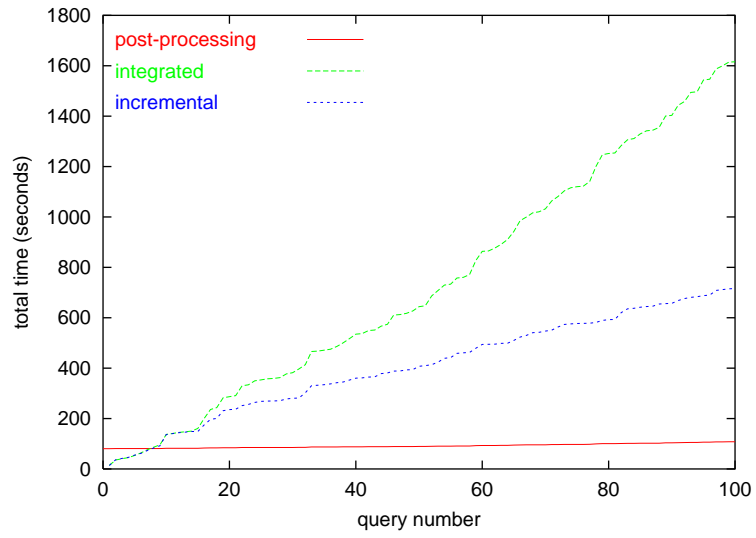
For each data set described in Section 4 we experimented with a session of 100 queries using the integrated querying approach, the post-processing approach and the incremental approach. Again, the queries used for the sessions were randomly generated. Figure 3 shows the evolution of the sessions in time. For all four sessions, the cut-off point where the integrated querying approach loses against the post-processing approach is the same for the incremental querying approach since not enough itemsets could be reused before that. Except for the mushroom data set, the incremental approach starts paying off after the twentieth query. Nevertheless, the reuse of previous results does not improve the performance enough for the incremental approach. Indeed, the incremental approach will always need some time to fetch all pre-generated itemsets and it will try to generate some more. However, as can be seen, the incremental approach shows a significant improvement on the integrated querying approach. Only for the mushroom data set, the cut-off point occurs at the fifth query, and almost all itemsets have been generated after the eighteenth query. As can be seen, the performance of the post-processing approach is very good compared to the other approaches. Nevertheless, if we still lowered the support thresholds, the post-processing approach becomes unfeasible to begin with, due to an overload of frequent itemsets. In that case, the integrated and incremental approach are still feasible and perform very similar as in the presented experiments.

6 Conclusions

This study revealed several insights into the association rule mining problem. First, due to recent advances on association rule mining algorithms, the performance has been significantly improved, such that the advantages of integrating constraints into the mining algorithm suddenly become less clear. Indeed, we showed that as long mining without any constraints is feasible, that is, if the number of frequent itemsets does not reach a huge amount, the total time spent to query the frequent itemsets and confident association rules becomes less after a certain amount of queries, compared to integrated querying, in which every

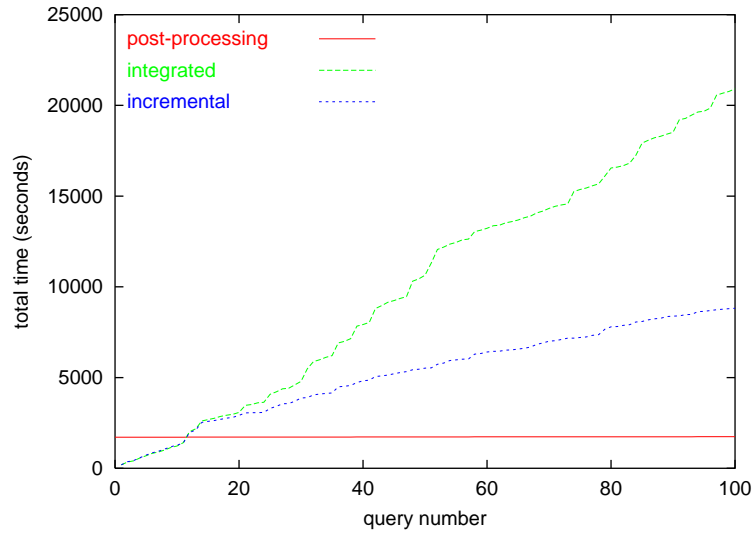


(a) basket

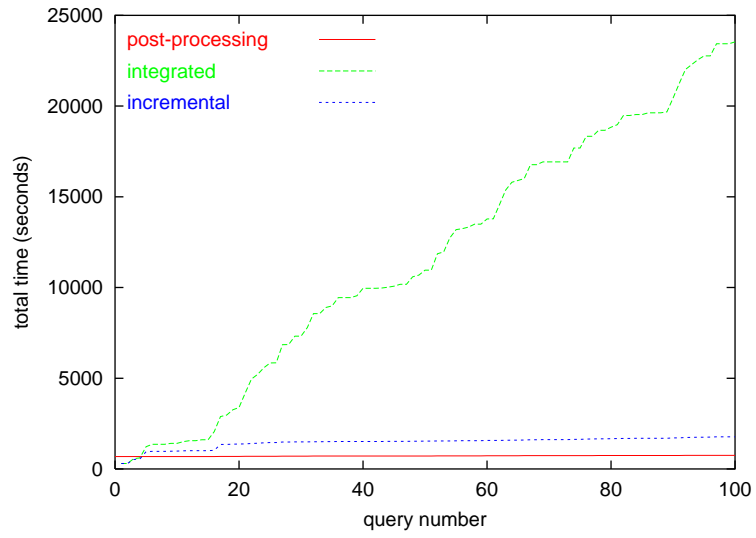


(b) BMS-Webview-1

Figure 3: Actual and estimated number of candidate patterns.



(c) T40I10D100K



(d) mushroom

Figure 3: Actual and estimated number of candidate patterns.

query is pushed into the mining algorithm. The incremental approach still improves the integrated approach by reusing as much previously generated results as possible. If the cut-off point would lie beyond the number of queries in which the user is interested, the incremental approach is obviously the best choice to use.

Of course, if the user is interested in some frequent itemsets and association rules which have very low frequencies, and hence mining without any constraints becomes infeasible, the incremental approach can still be performed.

Also note, that if a user is still interested in all frequent sets and association rules, but mining without constraints is infeasible, our queries can be used to divide the task over several runs, without spending much more time. For example, one can ask different queries of which the disjunction still gives all sets and rules. Essentially, this technique forms the basis of the well known Eclat [27] and FP-growth algorithms [10].

Acknowledgement

We wish to thank Blue Martini Software for contributing the KDD Cup 2000 data, the machine learning repository librarians Catherine Blake and Chris Mertz for providing access to the mushroom data, and Tom Brijs for providing the Belgian retail market basket data.

References

- [1] R. Agrawal, T. Imielinski, and A.N. Swami. Mining association rules between sets of items in large databases. In P. Buneman and S. Jajodia, editors, *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, volume 22:2 of *SIGMOD Record*, pages 207–216. ACM Press, 1993.
- [2] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A.I. Verkamo. Fast discovery of association rules. In Fayyad et al. [6], pages 307–328.
- [3] R. Agrawal and R. Srikant. *Quest Synthetic Data Generator*. IBM Almaden Research Center, <http://www.almaden.ibm.com/cs/quest/syndata.html>.
- [4] C.L. Blake and C.J. Merz. *UCI Repository of machine learning databases*. University of California, Irvine, Dept. of Information and Computer Sciences, <http://www.ics.uci.edu/~mlearn/MLRepository.html>, 1998.
- [5] U.M. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. From data mining to knowledge discovery: An overview. In Fayyad et al. [6], pages 1–34.
- [6] U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors. *Advances in Knowledge Discovery and Data Mining*. MIT Press, 1996.

- [7] G. Grahne, L.V.S. Lakshmanan, and X. Wang. Efficient mining of constrained correlated sets. In *Proceedings of the 16th International Conference on Data Engineering*, pages 512–521. IEEE Computer Society, 2000.
- [8] J. Han, Y. Fu, K. Koperski, W. Wang, and O. Zaiane. DMQL: A data mining query language for relational databases. Presented at SIGMOD’96 Workshop on Research Issues on Data Mining and Knowledge Discovery, 1996.
- [9] J. Han, Y. Fu, W. Wang, et al. DBMiner: A system for mining knowledge in large relational databases. In E. Simoudis, J. Han, and U. Fayyad, editors, *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, pages 250–255. AAAI Press, 1996.
- [10] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In W. Chen, J.F. Naughton, and P.A. Bernstein, editors, *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, volume 29:2 of *SIGMOD Record*, pages 1–12. ACM Press, 2000.
- [11] D. Heckerman, H. Mannila, and D. Pregibon, editors. *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining*. AAAI Press, 1997.
- [12] T. Imielinski and H. Mannila. A database perspective on knowledge discovery. *Communications of the ACM*, 39(11):58–64, 1996.
- [13] T. Imielinski and A. Virmani. MSQL: A query language for database mining. *Data Mining and Knowledge Discovery*, 3(4):373–408, December 1999.
- [14] B. Jeudy and J-F. Boulicaut. Using condensed representations for interactive association rule mining. In T. Elomaa, H. Mannila, and H. Toivonen, editors, *Proceedings of the 6th European Conference on Principles of Data Mining and Knowledge Discovery*, volume 2431 of *Lecture Notes in Computer Science*, pages 225–236. Springer, 2002.
- [15] R.J. Bayardo Jr., R. Agrawal, and D. Gunopulos. Constraint-based rule mining on large, dense data sets. In *Proceedings of the 15th International Conference on Data Engineering*, pages 188–197. IEEE Computer Society, 1999.
- [16] Y. Kambayashi, M.K. Mohania, and A.M. Tjoa, editors. *Proceedings of the Second International Conference on Data Warehousing and Knowledge Discovery*, volume 1874 of *Lecture Notes in Computer Science*. Springer, 2000.
- [17] R. Kohavi, C. Brodley, B. Frasca, L. Mason, and Z. Zheng. KDD-Cup 2000 organizers’ report: Peeling the onion. *SIGKDD Explorations*, 2(2):86–98, 2000. <http://www.ecn.purdue.edu/KDDCUP>.

- [18] L.V.S. Lakshmanan, R.T. Ng, J. Han, and A. Pang. Optimization of constrained frequent set queries with 2-variable constraints. In A. Delis, C. Faloutsos, and S. Ghandeharizadeh, editors, *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, volume 28:2 of *SIGMOD Record*, pages 157–168. ACM Press, 1999.
- [19] R. Meo, G. Psaila, and S. Ceri. A new SQL-like operator for mining association rules. In T.M. Vijayaraman, A.P. Buchmann, C. Mohan, and N.L. Sarda, editors, *Proceedings 22nd International Conference on Very Large Data Bases*, pages 122–133. Morgan Kaufmann, 1996.
- [20] Biswadeep Nag, Prasad Deshpande, and David J. DeWitt. Using a knowledge cache for interactive discovery of association rules. In U. Fayyad, S. Chaudhuri, and D. Madigan, editors, *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 244–253. ACM Press, 1999.
- [21] R.T. Ng, L.V.S. Lakshmanan, J. Han, and A. Pang. Exploratory mining and pruning optimizations of constrained association rules. In L.M. Haas and A. Tiwary, editors, *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, volume 27:2 of *SIGMOD Record*, pages 13–24. ACM Press, 1998.
- [22] J. Pei and J. Han. Can we push more constraints into frequent pattern mining? In R. Ramakrishnan, S. Stolfo, R. Bayardo, and I. Parsa, editors, *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 350–354. ACM Press, 2000.
- [23] J. Pei, J. Han, and L.V.S. Lakshmanan. Mining frequent itemsets with convertible constraints. In *Proceedings of the 17th International Conference on Data Engineering*, pages 433–442. IEEE Computer Society, 2001.
- [24] R. Srikant and R. Agrawal. Mining generalized association rules. In U. Dayal, P.M.D. Gray, and S. Nishio, editors, *Proceedings 21th International Conference on Very Large Data Bases*, pages 407–419. Morgan Kaufmann, 1995.
- [25] R. Srikant, Q. Vu, and R. Agrawal. Mining association rules with item constraints. In Heckerman et al. [11], pages 66–73.
- [26] H. Toivonen. Sampling large databases for association rules. In T. M. Vijayaraman, Alejandro P. Buchmann, C. Mohan, and Nandlal L. Sarda, editors, *Proceedings 22th International Conference on Very Large Data Bases*, pages 134–145. Kaufmann, 1996.
- [27] M.J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. New algorithms for fast discovery of association rules. In Heckerman et al. [11], pages 283–296.