On Supporting Interactive Association Rule Mining

Bart Goethals and Jan Van den Bussche

Limburg University, Belgium
http://www.luc.ac.be/theocomp/

Abstract. We investigate ways to support interactive mining sessions, in the setting of association rule mining. In such sessions, users specify conditions (filters) on the associations to be generated. Our approach is a combination of the incorporation of filtering conditions inside the mining phase, and the filtering 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 [3]. It motivated the idea of a "data mining query language" [5–8, 10] and was stressed again by Ng, Lakshmanan, Han and Pang [11]. A data mining query language allows the user to ask for specific subsets of association rules. Efficiently supporting data mining query language environments is a challenging task.

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*).

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. The filtering achieved by exploiting the query conditions is *non-redundant*, in the sense that it never generates an itemset that, apart from the minimal support and confidence thresholds, could give rise to a rule that does not satisfy the query. Therefore, the number of generated itemsets during the execution of a query, becomes proportional to the strength of the conditions in the query: the more specific the query, the faster its execution.
2. 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 conditions in the query.

3. 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.

The idea that filters can be integrated in the mining algorithm was initially launched by Srikant, Vu, and Agrawal [12], who considered filters that are Boolean expressions over the presence or absence of certain items in the rules (filters specifically as bodies or heads were not discussed). The algorithms proposed in their paper are not optimal: they generate and test several itemsets that do not satisfy the filter, and their optimizations also do not always become more efficient for more specific filters.

Also Lakshmanan, Ng, Han and Pang worked on the integration of constraints on itemsets in mining, considering conjunctions of conditions such as those considered here, as well as others (arbitrary Boolean combinations were not discussed) [9, 11]. Of the various strategies for the so-called "CAP" algorithm they present, the one that can handle the filters considered in the present paper is their "strategy II." Again, this strategy generates and tests itemsets that do not satisfy the filter. Also their algorithms implement a rule-filter by separately mining for possible heads and for possible bodies, while we tightly couple filtering of rules with filtering of sets.

Both works do not discuss the reuse of results acquired from earlier queries within a session.

This paper is further organized as follows. We assume familiarity with the notions and terminology of association rule mining and the Apriori algorithm [1, 2]. In Section 2, we present a way of incorporating query-constraints inside a frequent set mining algorithm. In Section 3, we discuss ways of supporting interactive mining sessions. In Section 4, we present an experimental evaluation of our algorithms, and discuss their implementation.

2 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 Body(i) or 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.

2.1 Conjunctive Constraints

Let $b_1, \ldots, b_\ell$ be the items that must be in the body by the constraint; $b'_1, \ldots, b'_{\ell'}$ those that must not; $h_1, \ldots, h_m$ those that must be in the head; and $h'_1, \ldots, h'_{m'}$ 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-filter corresponding to the rule-filter, which is also a conjunctive expression, but now over the presence or absence of an item i in a frequent set, denoted by $\mathrm{Set}(i)$ and $\neg\mathrm{Set}(i)$. We do this as follows:

1. For each positive literal $\mathrm{Body}(i)$ or $\mathrm{Head}(i)$ in the rule-filter, add the literal $\mathrm{Set}(i)$ in the set-filter.
2. If for an item i both $\neg\mathrm{Body}(i)$ and $\neg\mathrm{Head}(i)$ are in the rule-filter, add the negated literal $\neg\mathrm{Set}(i)$ to the set-filter.
3. Add the minimal support threshold to the set-filter.
4. All other literals in the rule-filter 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-filter 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-filter, apart from the confidence threshold.*

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

We thus need to explain two things:

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

Finding the frequent sets satisfying the set-filter. Let $Pos := \{i \mid \mathrm{Set}(i)$ in set-filter$\}$ and $Neg := \{i \mid \neg\mathrm{Set}(i)$ in set-filter$\}$. Note that $Pos = \{b_1, \ldots, b_\ell, h_1, \ldots, h_m\}$. 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: (proof omitted)

Lemma 2. *Let p be the absolute support threshold defined in the filter. 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-filter, and whose support is also at least p. Then*

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

We can thus perform any frequent set generation algorithm, using only $\mathcal{D}_0$ instead of $\mathcal{D}$. Note that the size of $\mathcal{D}_0$ is exactly the support of Pos in $\mathcal{D}$. Still put differently: we are mining in a world where itemsets that do not satisfy the filter 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-filtered 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-filter can be generated. Recall that in phase 2 of Apriori, 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 filter, a head must always be a superset of $\{h_1, \ldots, h_m\}$ and must not include any of the h'_i and b_i (the latter because bodies and heads of rules are disjoint). In this way, we again head-start with a lead of m. Similarly, a body must always be a superset of $\{b_1, \ldots, b_\ell\}$ and may not include any of the b'_i and h_i.

The following lemma 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-filter. Then*

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

and

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

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 in one additional pass. (We do not yet have these frequencies because these sets do not contain either items b_i or h_i, while we ignored transactions that do 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-filter;
2. A frequent set that can act as body of a rule that satisfies the rule-filter;
3. Analogously for a head.

We finally generate the desired association rules from the appropriate sets, on condition that they have enough confidence.

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

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

We begin by converting it to the set-filter

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

Hence $Pos = \{1, 3\}$ and $Neg = \{5\}$. 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. After reading, according to Lemmas 1 and 2, the two resulting transactions, one of the itemsets we find in $\mathcal{S}_0$ is $\{4, 8\}$, which 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-filter. 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.

2.2 Arbitrary Boolean Filters

Assume now given a rule-filter that is an arbitrary Boolean combination of atomic conditions. We can put it in Disjunctive Normal Form[1] and then generate all frequent itemsets for every disjunct (which is a conjunction) in parallel by 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 filter: $\mathrm{Body}(1) \vee \mathrm{Body}(2)$. If we convert it to its corresponding set-filter (disjunct by disjunct), we get $\mathrm{Set}(1) \vee \mathrm{Set}(2)$. Then, we would generate for both disjuncts all supersets of $\{1, 2\}$. We can avoid this problem by putting the set-filter in *disjoint* DNF.[2] 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-filter: $\mathrm{Set}(1) \vee (\mathrm{Set}(2) \wedge \neg\mathrm{Set}(1))$. 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-filter in disjoint DNF.

[1] Any Boolean expression has an equivalent DNF.

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

Until now, we have disregarded the possible presence of negated thresholds in the filters, which can come from the conversion to disjoint DNF, or from the user himself. Due to space limitations, we defer their treatment to the full paper.

3 Interactive Mining

3.1 Integrated Filtering or Post-Processing?

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

Integrated filtering has obvious advantages over post-processing:

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-filtered mining operation, on the result of which the filtering will be performed by post-processing, *can be executed once and its result materialized for the remainder of the data mining session* (or part of it).

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 an integrated filter will involve at least 20, and often many more, passes over the data, as each query involves a separate mining operation. We can analyze the situation easily as follows.

Consider a session in which the user issues a total of m data mining queries over a database of size n. Suppose that the total number of association rules (given a minimal support and confidence requirement) over these data equals r. Let t be the time required to generate all these rules. Moreover, it is not unreasonable to estimate that in post-processing, each filter executes in time proportional to r, and that in integrated filtering, each filter executes in time proportional to n. Then the total time spent by the post-processing approach is $t + m \cdot r$, while in the integrated filtering approach this is $m \cdot n$. Hence, if $n > r$, we have proved the following:

Proposition 1. *The integrated filtering total time is guaranteed to grow beyond the post-processing total time after exactly* $m = \lceil t/(n-r) \rceil$ *queries.*

3.2 Online Filtering: Basic Approach

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

In the online approach, all rules and itemsets that result from every posed query, as well as all intermediate generated itemsets, are saved incrementally. Initially, when the user issues his first query, nothing has been mined yet, and thus we answer it using integrated filtering.

Every subsequent query is first converted to its corresponding rule- and set-filter in disjoint DNF. For every disjunct in the set-filter, the system checks all currently saved itemsets. If an itemset satisfies the disjunct, it is added 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 saved; if they are not, we have to count their supports during the first scan through the dataset). We also immediately add all candidate sets, and if they were already saved, we add their support, so that they need not to be regenerated and recounted.

If no new candidate sets can be generated, this means that all necessary sets were already saved, and we are done. However, if this is not the case, we can now begin our filtered 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 non-frequent must be kept so that they are not regenerated in later iterations. This generalization was first used by Toivonen in his sampling algorithm [13].

Saving all generated itemsets and rules gives us another advantage that can be exploited by the integrated filtering algorithm. Consider a set-filter stating that item 1 and 2 must be in the set. In the first pass of the algorithm all single itemsets are generated as candidate sets over the new dataset $\mathcal{D}_0$ (cf. Section 2.1). We explained that these single itemsets actually represent supersets of $\{1, 2\}$. Normally, before we generate a candidate set, 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 saved results for a subset with too low support; if we find this, we can avoid generating the candidate.

For rule generation, the same techniques apply. We thus obtain an algorithm which reuses previously generated itemsets and rules as if they had been generated in previous iterations of the algorithm. We are optimal in the sense that we never generate and test sets or rules that were generated before.

We also note that techniques for dealing with main memory overflow in the setting of standard, non-filtered, non-interactive association rule mining [14], remain valid in our approach.

3.3 Online Filtering: Improvements

In the worst case, the saved results do not contain anything that can be reused for answering a filter, and hence the time needed to generate the rules that satisfy the filter is equal to the time needed when answering that filter using the integrated filtering approach. In the best case, all requested rules are already saved, and hence the time needed to find all rules that satisfy the filter is equal to the time needed for answering that filter using post-processing. In the average case, part of the needed sets and rules are saved and will then be used to speed

up the integrated filtering approach. If the time gained by this speedup is more than the time needed to find the reusable sets and rules, then the online approach will always be faster than the integrated filtering approach. In the limit, all rules will be materialized, and hence all subsequent filters will be answered using post-processing.

Could it be that the time gained by the speedup in the integrated filtering is less than the time needed to find the reusable sets and rules? This could happen when a lot of sets and rules are already saved, but almost none of them satisfies the filter. We can however counter this phenomenon by improving the speedup. The improvement is based on estimating what is currently saved, as follows.

We keep track of a set-filter ϕ_{sets} which describes the saved sets, and of a rule-filter ψ_{rules} which describes the saved rules. Both filters are initially *false*. Given a new query (rule-filter) ψ the system now goes through the following steps: (step 3 was described in section 2.1)

1. $\psi_{mine} := \psi \wedge \neg\psi_{rules}$
2. $\psi_{rules} := \psi_{rules} \vee \psi$
3. Convert the rule-filter ψ_{mine} to the set-filter ϕ
4. $\phi_{mine} := \phi \wedge \neg\phi_{sets}$
5. $\phi_{sets} := \phi_{sets} \vee \phi$

After this, we perform:

1. Generate all frequent sets according to ϕ_{mine}, using the basic online approach.
2. Retrieve all saved sets satisfying $\phi \wedge \neg\phi_{mine}$.
3. Add all needed subsets that can serve as bodies or heads.
4. Use ψ_{mine} to generate rules from the sets of steps 1 and 2.
5. Retrieve all saved rules satisfying ψ.

Note that the filter ϕ_{mine} is much more specific than the original filter ϕ. We thus obtain the improvement in speedup from integrated filtering, which we already pointed out to be proportional to the strength of the filter.

3.4 Avoiding Exploding Filters

The improvement just described incurs a new problem. The formula ψ_{rules} (or ϕ_{sets}) becomes longer with the session. When, given the next filter ψ, we mine for $\psi \wedge \neg\psi_{rules}$, we will convert to disjoint DNF which could explode.

To avoid this, consider ψ_{rules} in DNF: $\psi_1 \vee \cdots \vee \psi_n$. Instead of the full filter $\psi \wedge \neg\psi_{rules}$, we are going to use a filter $\psi \wedge \neg\psi'_{rules}$, where ψ'_{rules} is obtained from ψ_{rules} by keeping only the least restrictive disjuncts ψ_i (their negation will thus be most restrictive). In this way $\psi \wedge \neg\psi'_{rules}$ is kept short.

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

4 Experimental Comparison

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

We experimented with a session of 40 queries using the integrated filtering approach, the post-processing approach and the online approach. The transaction database was synthetically generated using the program provided by the Quest research group at IBM Almaden and contained 1 000 000 transaction over 10 000 items. The performance figures are shown in Figure 1.

The first 20 queries all require different items in the rules such that the online approach is just a little faster than the integrated filtering approach, because it cannot reuse that much pre-generated sets and rules. From there on the online approach has already collected a lot of sets and rules that can be reused. After the 20th query we can see some improvement on this until the 30th query, where the online approach has collected all sets and rules needed to answer further queries and hence the time needed to answer these queries is equal to the time needed to answer these queries using the post-processing approach.

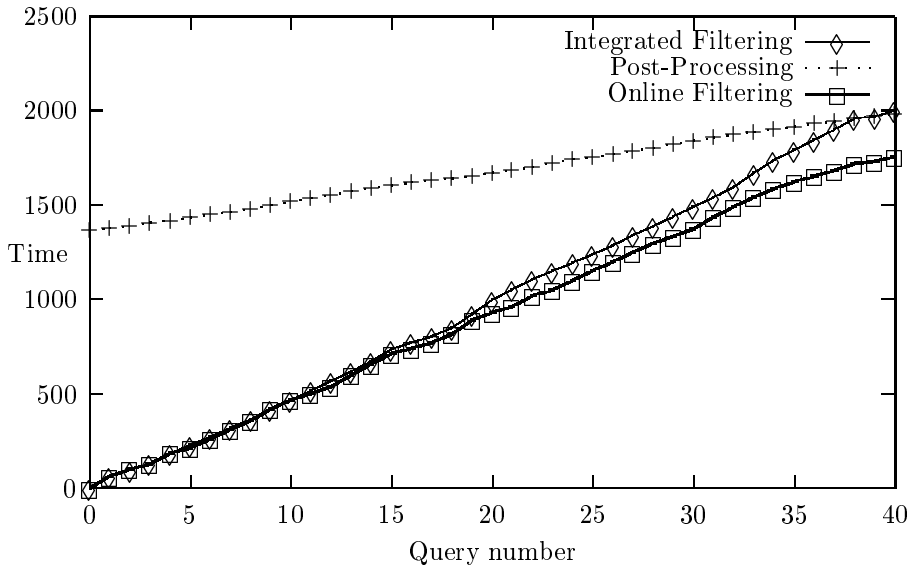

Fig. 1. Experiments. Time is in seconds.

References

1. R. Agrawal, T. Imielinski, and A.N. Swami. Mining association rules between sets of items in large databases. In *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. [4], pages 307–328.
3. U.M. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. From data mining to knowledge discovery: An overview. In Fayyad et al. [4], pages 1–34.
4. U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors. *Advances in Knowledge Discovery and Data Mining*. MIT Press, 1996.
5. 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.
6. 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 2nd International Conference on Knowledge Discovery & Data Mining*, pages 250–255. AAAI Press, 1996.
7. T. Imielinski and H. Mannila. A database perspective on knowledge discovery. *Communications of the ACM*, 39(11):58–64, 1996.
8. T. Imielinski and A. Virmani. MSQL: A query language for database mining. *Data Mining and Knowledge Discovery*, 3(4):373–408, December 1999.
9. 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.
10. R. Meo, G. Psaila, and S. Ceri. An extension to SQL for mining association rules. *Data Mining and Knowledge Discovery*, 2(2):195–224, June 1998.
11. 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.
12. R. Srikant, Q. Vu, and R. Agrawal. Mining association rules with item constraints. In D. Heckerman, H. Mannila, and D. Pregibon, editors, *Proceedings 3rd International Conference on Knowledge Discovery & Data Mining*, pages 66–73. AAAI Press, 1997.
13. 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.
14. Y. Xiao and M.H. Dunham. Considering main memory in mining association rules. In M. K. Mohania and A. Min Tjoa, editors, *Data Warehousing and Knowledge Discovery*, volume 1676 of *Lecture Notes in Computer Science*, pages 209–218. Springer, 1999.