Appears in Proc. of the 15th Int'l Conf. on Data Engineering, 188-197, 1999.

Constraint-Based Rule Mining in Large, Dense Databases

Roberto J. Bayardo Jr.
IBM Almaden Research Center
bayardo@alum.mit.edu

Rakesh Agrawal
IBM Almaden Research Center
rakesh_agrawal@ieee.org

Dimitrios Gunopulos
IBM Almaden Research Center
dg@cs.ucr.edu

Abstract

Constraint-based rule miners find all rules in a given data-set meeting user-specified constraints such as minimum support and confidence. We describe a new algorithm that directly exploits all user-specified constraints including minimum support, minimum confidence, and a new constraint that ensures every mined rule offers a predictive advantage over any of its simplifications. Our algorithm maintains efficiency even at low supports on data that is dense (e.g. relational data). Previous approaches such as Apriori and its variants exploit only the minimum support constraint, and as a result are ineffective on dense data due to a combinatorial explosion of "frequent itemsets".

1. Introduction

Mining rules from data is a problem that has attracted considerable interest because a rule provides a concise statement of potentially useful information that is easily understood by end users. In the database literature, the focus has been on developing association rule [1] algorithms that identify all conjunctive rules meeting user-specified constraints such as minimum support (a statement of generality) and minimum confidence (a statement of predictive ability). These algorithms were initially developed to tackle data-sets primarily from the domain of market-basket analysis, though there has been recent interest in applying these algorithms to other domains including telecommunications data analysis [14], census data analysis [6], and classification and predictive modeling tasks in general. Unlike data from market-basket analysis, these data-sets tend to be *dense* in that they have any or all of the following properties:[1]

- many frequently occurring items (e.g. sex=male);
- strong correlations between several items;
- many items in each record.

These data-sets can cause an exponential blow-up in the resource consumption of standard association rule mining algorithms including Apriori [2] and its many variants. The combinatorial explosion is a result of the fact that these algorithms effectively mine all rules that satisfy only the minimum support constraint, the number of which is exorbitant. Though other rule constraints are specifiable, they are typically enforced solely during a post-processing filter step. Our approach to mining on dense data-sets is to instead directly enforce all user-specified rule constraints during mining. For example, most association rule miners allow users to set a minimum on the predictive ability of any mined rule specified as either a minimum confidence [1] or an alternative measure such as lift [5,8] (also known as interest [6]) or conviction [6]. Our algorithm can exploit such minimums on predictive ability during mining for vastly improved efficiency.

Even given strong minimums on support and predictive ability, the rules satisfying these constraints in a dense data-set are often too numerous to be mined efficiently or comprehended by the end user. To remedy this problem, our algorithm exploits another constraint that eliminates rules that are uninteresting because they contain conditions that do not (strongly) contribute to the predictive ability of the rule. To illustrate this useful concept, first consider the rule below:

`Bread & Butter` $\rightarrow$ `Milk (Confidence = 80%)`

This rule has a confidence of 80%, which says that 80% of the people who purchase bread and butter also purchase the item in the *consequent* of the rule, which is milk. Because of its high confidence, one might be inclined to believe that this rule is an interesting finding if the goal is to, say, understand the population of likely milk buyers in order to make better stocking and discounting decisions. However, if 85% of the population under examination purchased milk, this rule is actually quite uninteresting for this purpose since it characterizes a population that is even less likely to buy milk than the average shopper. This point has motivated additional measures for identifying interesting rules including lift and conviction. Both lift and conviction represent the predictive advantage a rule offers over simply guessing based on the frequency of the consequent. But both measures still exhibit another closely related problem illustrated by the next rule.

`Eggs & Cereal` $\rightarrow$ `Milk (Confidence = 95%)`

Because the confidence of this rule (95%) is significantly higher than the frequency with which milk is purchased (85%), this rule will have lift and conviction values that could imply to the end-user that it is interesting for understanding likely milk buyers. But suppose the purchase of cereal alone implies that milk is purchased with 99% confidence. We then have that the above rule actually represents

[1] Market-basket data is sometimes dense, particularly when it incorporates information culled from convenience card applications for mining rules that intermix personal attributes with items purchased.

a significant decrease in predictive ability over a more concise rule which is more broadly applicable (because there are more people who buy cereal than people who buy both cereal and eggs).

To address these problems, our algorithm allows the user to specify a *minimum improvement* constraint. The idea is to mine only those rules whose confidence is at least *minimp* greater than the confidence of *any* of its proper sub-rules, where a proper sub-rule is a simplification of the rule formed by removing one or more conditions from its antecedent. Any positive setting of minimp would prevent the undesirable rules from the examples above from being generated by our algorithm. More generally, the minimum improvement constraint remedies the rule explosion problem resulting from the fact that in dense data-sets, the confidence of many rules can often be marginally improved upon in an overwhelming number of ways by adding additional conditions. For example, given the rule stating that cereal implies milk with 99% confidence, there may be hundreds of rules of the form below with a confidence between 99% and 99.1%.

$$\texttt{Cereal \& } I_1 \texttt{ \& } I_2 \texttt{ \& } \ldots \texttt{ \& } I_n \rightarrow \texttt{Milk}$$

By specifying a small positive value for minimp, one can trade away such marginal benefits in predictive ability for a far more concise set of rules, with the added property that every returned rule consists entirely of items that are strong contributors to its predictive ability. We feel this is a worthwhile trade-off in most situations where the mined rules are used for end-user understanding.

For rules to be comparable in the above-described context, they must have equivalent consequents. For this reason, our work is done in the setting where the consequent of the rules is fixed and specified in advance. This setting is quite natural in many applications where the goal is to discover properties of a specific class of interest. This task is sometimes referred to as partial-classification [3]. Some example domains where it is applicable include failure analysis, fraud detection, and targeted marketing among many others.

1.1 Paper overview

Section 2 summarizes related work. Section 3 formally defines and motivates the problem of mining rules from dense data subject to minimum support, confidence, and/or improvement constraints. The next three sections define our algorithm in a top-down manner. Section 4 begins with an overview of the general search strategy, and presents pseudo-code for the top level of our algorithm. Section 5 provides details and pseudo-code for the pruning functions invoked by the algorithm body. Section 6 details the item-reordering heuristic, and Section 7 describes the rule post-processor. The algorithm is empirically evaluated in Section 8. Section 9 concludes with a summary of the contributions.

2. Related work

Previous work on mining rules from data is extensive. We will not review the numerous proposals for greedy or heuristic rule mining (e.g. decision tree induction) and focus instead on algorithms that provide completeness guarantees. We refer the reader interested in heuristic approaches for mining large data-sets to the scalable algorithms proposed in [7] and [12].

There are numerous papers presenting improvements to the manner in which the Apriori algorithm [2] enumerates all frequent itemsets (e.g. [6]), though none address the problem of combinatorial explosion in the number of frequent itemsets resulting from applying these techniques to dense data. Other works (e.g. [4]) show how to identify all maximal frequent itemsets in data-sets where the frequent itemsets are long and numerous. Unfortunately, all association rules cannot be efficiently extracted from maximal frequent itemsets alone, as this would require performing the intractable task of enumerating and computing the support of all their subsets.

Srikant et al. [14] and Ng et al. [10] have investigated incorporating item constraints on the set of frequent itemsets for faster association rule mining. These constraints, which restrict the items or combinations of items that are allowed to participate in mined rules, are orthogonal to those exploited by our approach. We believe both classes of constraints should be part of any rule-mining tool or application.

There is some work on ranking association rules using interest measures [6,8,9], though this work gives no indication of how these measures could be exploited to make mining on dense data-sets feasible. Smythe and Goodman [13] describe a constraint-based rule miner that exploits an information theoretic constraint which heavily penalizes long rules in order to control model and search complexity. We incorporate constraints whose effects are more easily understood by the end user, and allow efficient mining of long rules should they satisfy these constraints.

There are several proposals for constraint-based rule mining with a machine-learning instead of data-mining focus that do not address the issue of efficiently dealing with large data-sets. Webb [15] provides a good survey of this class of algorithms, and presents the OPUS framework which extends the set-enumeration search framework of Rymon [11] with additional generic pruning methods. Webb instantiates his framework to produce an algorithm for obtaining a single rule that is optimal with respect to the Laplace preference function. We borrow from this work the idea of exploiting an optimistic pruning function in the context of lattice-space search. However, instead of using a single pruning function for optimization, we use several for constraint enforcement. Also, because the itemset frequency information required for exploiting pruning functions is expensive to obtain from a large data-set, we frame our pruning functions so that they can accommodate restricted availability of such information.

3. Definitions and problem statement

A *transaction* is a set of one or more items obtained from a finite item domain, and a *data-set* is a collection of transactions. A set of items will be referred to more succinctly as an *itemset*. The *support* of an itemset I, denoted $\sup(I)$, is

the number of transactions in the data-set to contain I. An *association rule*, or just *rule* for short, consists of an itemset called the antecedent, and an itemset disjoint from the antecedent called the *consequent*. A rule is denoted as $A \rightarrow C$ where A is the antecedent and C the consequent. The *support* of an association rule is the support of the itemset formed by taking the union of the antecedent and consequent ($A \cup C$). The *confidence* of an association rule is the probability with which the items in the antecedent A appear together with items in the consequent C in the given data-set. More specifically:

$$\text{conf}(A \rightarrow C) = \frac{\text{sup}(A \cup C)}{\text{sup}(A)}$$

Other measures of predictive ability include *lift* [5,8], which is also known as *interest* [6], and conviction [6]. The conviction and lift of a rule can each be expressed as a function of the rule's confidence and the frequency of the consequent; further, both functions are monotone in confidence:

$$\text{lift}(A \rightarrow C) = \frac{\text{sup}(\varnothing)}{\text{sup}(C)} \cdot \text{conf}(A \rightarrow C)$$

$$\text{conviction}(A \rightarrow C) = \frac{\text{sup}(\varnothing) - \text{sup}(C)}{\text{sup}(\varnothing)[1 - \text{conf}(A \rightarrow C)]}$$

Though we frame the remainder of this work in terms of confidence, it can be easily recast in terms of any measure with this monotonicity property.

The association rule mining problem [1] is to produce all association rules present in a data-set that meet specified minimums on support and confidence. In this paper, we restrict the problem in two ways in order to render it solvable given dense data.

3.1 The consequent constraint

We require mined rules to have a given consequent C specified by the user. This restriction is an *item constraint* which can be exploited by other proposals [10, 14], but only to reduce the set of frequent itemsets considered. A frequent itemset is a set of items whose support exceeds the minimum support threshold. Frequent itemsets are too numerous in dense data even given this item constraint. Our algorithm instead leverages the consequent constraint through pruning functions for enforcing confidence, support, and improvement (defined next) constraints during the mining phase.

3.2 The minimum improvement constraint

While our algorithm runs efficiently on many dense data-sets without further restriction, the end-result can easily be many thousands of rules, with no indication of which ones are "good". On some highly dense data-sets, the number of rules returned explodes as support is decreased, resulting in unacceptable algorithm performance and a rule-set the end-user has no possibility of digesting. We address this problem by introducing an additional constraint.

Let the *improvement* of a rule be defined as the minimum difference between its confidence and the confidence of any proper sub-rule with the same consequent. More formally, given a rule $A \rightarrow C$:

$$\text{imp}(A \rightarrow C) = \min(\forall A' \subset A, \text{conf}(A \rightarrow C) - \text{conf}(A' \rightarrow C))$$

If the improvement of a rule is positive, then removing any non-empty combination of items from its antecedent will drop its confidence by at least its improvement. Thus, every item and every combination of items present in the antecedent of a large-improvement rule is an important contributor to its predictive ability. A rule with negative improvement is typically undesirable because the rule can be simplified to yield a proper sub-rule that is more predictive, and applies to an equal or larger population due to the antecedent containment relationship. An improvement greater than 0 is thus a desirable constraint in almost any application of association rule mining. A larger minimum on improvement is also often justified because most rules in dense data-sets are not useful due to conditions or combinations of conditions that add only a marginal increase in confidence. Our algorithm allows the user to specify an arbitrary positive minimum on improvement.

3.3 Problem statement

We develop an algorithm for mining all association rules with consequent C meeting user-specified minimums on support, confidence, and improvement. The algorithm parameter specifying the minimum confidence bound is known as *minconf*, and the minimum support bound *minsup*. We call the parameter specifying a minimum bound on improvement *minimp*. A rule is said to be *confident* if its confidence is at least minconf, and *frequent* if its support is at least minsup. A rule is said to have a *large improvement* if its improvement is at least minimp.

4. Set-enumeration search in large data-sets

From now on, we will represent a rule using only its antecedent itemset since the consequent is assumed to be fixed to itemset C. Let U denote the set of all items present in the database except for those in the consequent. The rule-mining problem is then one of searching through the power set of U for rules which satisfy the minimum support, confidence, and improvement constraints. Rymon's set-enumeration tree framework [11] provides a scheme for representing a subset search problem as a tree search problem, allowing pruning rules to be defined in a straightforward manner in order to reduce the space of subsets (rules) considered. The idea is to first impose an ordering on the set of items, and then enumerate sets of items according to the ordering as illustrated in Figure 1.

FIGURE 1. A completely expanded set-enumeration tree over $U = \{1, 2, 3, 4\}$, with items ordered lexically.

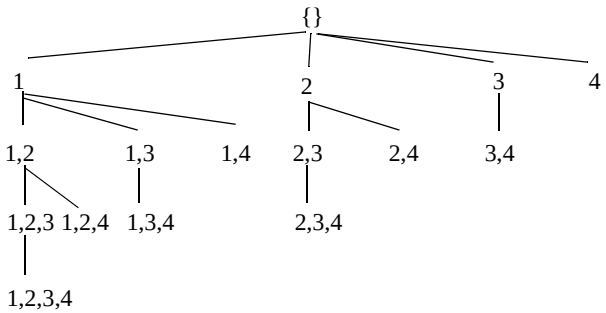


FIGURE 2. Dense-Miner at its top level. The input parameters minconf, minsup, minimp, and C are assumed global.

```
DENSE-MINER(Set of Transactions T)
    ;; Returns all frequent, confident, large
    ;; improvement rules present in T
    Set of Rules R ← ∅
    Set of Groups G ← GENERATE-INITIAL-GROUPS(T, R)
    while G is non-empty do
        scan T to process all groups in G
        PRUNE-GROUPS(G, R)                ;; Section 5
        G ← GENERATE-NEXT-LEVEL(G)
        R ← R ∪ EXTRACT-RULES(G)
        PRUNE-GROUPS(G, R)                ;; Section 5
    return POST-PROCESS(R, T)             ;; Section 7
```

4.1 Terminology

We draw upon the machinery developed in previous work where we framed the problem of mining maximal frequent itemsets from databases as a set-enumeration tree search problem [4]. Each node in the tree is represented by two itemsets called a *group*. The first itemset, called the *head*, is simply the itemset (rule) enumerated at the given node. The second itemset, called the *tail*, is actually an ordered set, and consists of those items which can be potentially appended to the head to form any viable rule enumerated by a sub-node. For example, at the root of the tree, the head itemset is empty and the tail itemset consists of all items in U.

The head and tail of a group g will be denoted as $h(g)$ and $t(g)$ respectively. The order in which tail items appear in $t(g)$ is significant since it reflects how its children are to be expanded (Figure 3). Each child g_c of a group g is formed by taking an item $i \in t(g)$ and appending it to $h(g)$ to form $h(g_c)$. Then, $t(g_c)$ is made to contain all items in $t(g)$ that follow i in the ordering. Given this child expansion policy, without any pruning of nodes or tail items, the set-enumeration tree enumerates each and every subset of U exactly once.

We say a rule r is *derivable* from a group g if $h(g) \subset r$, and $r \subseteq h(g) \cup t(g)$. By definition, any rule that can be enumerated by a descendent of g in the set-enumeration tree is also derivable from g.

Define the *candidate set* of a group g to be the set consisting of the following itemsets:

- $h(g)$ and $h(g) \cup C$;
- $h(g) \cup \{i\}$ and $h(g) \cup \{i\} \cup C$ for all $i \in t(g)$;
- $h(g) \cup t(g)$ and $h(g) \cup t(g) \cup C$.

A group is said to be *processed* once the algorithm has computed the support of every itemset in its candidate set.

4.2 Top-level algorithm description

It is now possible to provide a top-level description of the algorithm, which we call Dense-Miner. The body (Figure 2) implements a breadth-first search of the set enumeration tree with Generate-Initial-Groups seeding the search. The groups representing an entire level of the tree are processed together in one pass over the data-set. Though any systematic traversal of the set-enumeration tree could be used, Dense-Miner uses a breadth-first traversal to limit the number of database passes to at most the length of the longest frequent itemset. The use of hash-trees and other implementation details for efficient group processing is described in [4].

FIGURE 3. Procedure for expanding the next level of the set-enumeration tree.

```
GENERATE-NEXT-LEVEL(Set of groups G)
    ;; Returns a set of groups representing the next level
    ;; of the set-enumeration tree
    Set of Groups G_c ← ∅
    for each group g in G do
        reorder the items in t(g)          ;; Section 6
        for each item i in t(g) do
            let g_c be a new group
                with h(g_c) = h(g) ∪ {i} and
                t(g_c) = {j | j follows i in the ordering}
            G_c ← G_c ∪ {g_c}
    return G_c
```

Generate-Initial-Groups could simply produce the root node which consists of an empty head and U for its tail. However, our implementation seeds the search at the second level of the tree after an optimized phase that rapidly computes the support of all 1 and 2 item rules and their antecedents using array data-structures instead of hash trees.

Generate-Next-Level (Figure 3) generates the groups that comprise the next level of the set-enumeration tree. Note that the tail items of a group are reordered before its children are expanded. This reordering step is a crucial optimization designed to maximize pruning efficiency. We delay discussing the details of item reordering until after the pruning strategies are described, because the particular pruning operations greatly influence the reordering strategy. After child expansion, any rule represented by the head of a group is placed into R by Extract-Rules if it is frequent and confident. The support information required to check if the head of a group g represents a frequent or confident rule is provided by the parent of g in the set-enumeration tree because $h(g)$ and $h(g) \cup C$ are members of its candidate set. As a result, this step can be performed before g is processed.

The remaining algorithmic details, which include node pruning (the PRUNE-GROUPS function), item-reordering, and post-processing (the POST-PROCESS function), are the subjects of the next three sections.

5. Pruning

This section describes how Dense-Miner prunes both processed and unprocessed groups. In Figure 2, note that groups are pruned following tree expansion as well as immediately after they are processed. Because groups are unprocessed following tree expansion, in order to determine if they are prunable, Dense-Miner uses support information gathered during previous database passes.

FIGURE 4. Top level of the pruning function.

```
PRUNE-GROUPS(Set of groups G, Set of rules R)
    ;; Prunes groups and tail items from groups within G
    ;; G and R are passed by reference
    for each group g in G do
            do
                try_again ← false
                if IS-PRUNABLE(g)
                  then remove g from G
                  else for each i ∈ t(g) do
                            let g' be a group
                              with h(g') = h(g) ∪ {i}
                              and t(g') = t(g) – {i}
                            if IS-PRUNABLE(g')
                              then remove i from t(g)
                                   put h(g) ∪ {i} in R if it
                                     is a frequent and
                                     confident rule
                                   try_again ← true
            while try_again = true
```

5.1 Applying the pruning strategies

Dense-Miner applies multiple strategies to prune nodes from the search tree. These strategies determine when a group g can be pruned because no derivable rule can satisfy one or more of the input constraints. When a group g cannot be pruned, the pruning function checks to see if it can instead prune some items i from $t(g)$. Pruning tail items reduces the number of children generated from a node, and thereby reduces the search space. An added benefit of pruning tail items is that it can increase the effectiveness of the strategies used for group pruning. The observation below, which follows immediately from the definitions, suggests how any method for pruning groups can also be used to prune tail items.

OBSERVATION 5.1: Given a group g and an item $i \in t(g)$, consider the group g' such that $h(g') = h(g) \cup \{i\}$ and $t(g') = t(g) - \{i\}$. If no rules derivable from g' satisfy some given constraints, then except for rule $h(g) \cup \{i\}$, no rules r derivable from g such that $i \in r$ satisfy the given constraints.

The implication of this fact is that given a group g and tail item i with the stated condition, we can avoid enumerating many rules which do not satisfy the constraints by simply removing i from $t(g)$ after extracting rule $h(g) \cup \{i\}$ if necessary. The implementation of Prune-Groups, described in Figure 4, exploits this fact.

The group pruning strategies are applied by the helper function Is-Prunable which is described next. Because fewer tail items can improve the ability of Is-Prunable to determine whether a group can be pruned, whenever a tail item is found to be prunable from a group, the group and all tail items are checked once more.

5.2 Pruning strategies

The function Is-Prunable computes the following values for the given group g:

- an upper-bound uconf(g) on the confidence of any rule derivable from g,
- an upper-bound uimp(g) on the improvement of any rule derivable from g that is frequent,
- an upper-bound usup(g) on the support of any rule derivable from g.

Note that a group g can be pruned without affecting the completeness of the search if one of the above bounds falls below its minimum allowed value as specified by minconf, minimp, and minsup respectively. The difficulty in implementing pruning is not simply how to compute these three bounds, but more specifically, how to compute them given that acquiring support information from a large data-set is time consuming. We show how to compute these bounds using only the support information provided by the candidate set of the group, and/or the candidate set of its parent.

In establishing these bounding techniques in the remaining sub-sections, we assume the existence of a special "derived" item $\neg c$ that is contained only by those transactions in the data-set that do not contain the consequent itemset C. Similarly, for a given item i, we sometimes assume the existence of an item $\neg i$ contained only by those transactions that do not contain i. These derived items need not actually be present in the data-set -- they are used only to simplify the presentation. For an itemset $I \cup \{\neg c\}$, we have that $\sup(I \cup \{\neg c\}) = \sup(I) - \sup(I \cup C)$. We also exploit the fact that $I_1 \subset I_2 \Rightarrow \sup(I_1) \geq \sup(I_2)$, which holds whether or not I_1 and/or I_2 contain derived items.

5.3 Bounding confidence

THEOREM 5.2: The following expression provides an upper-bound on the confidence of any rule derivable from a given group g:

$$\frac{x}{x+y}$$

where x and y are non-negative integers such that $y \leq \sup(h(g) \cup t(g) \cup \{\neg c\})$ and $x \geq \sup(h(g) \cup C)$.

Proof: Recall that the confidence of a rule r is equal to $\sup(r \cup C) / \sup(r)$. This fraction can be rewritten as follows:

$$\frac{x'}{x'+y'} \text{ where } x' = \sup(r \cup C) \text{ and } y' = \sup(r) - \sup(r \cup C).$$

Because this expression is monotone in x' and anti-monotone in y', we can replace x' with a greater or equal value and y' with a lesser or equal value without decreasing the value of the expression. Consider replacing x' with x and y' with y. The claim then follows if we establish that for any rule r derivable from g, (1) $x \geq x'$, and (2) $y \leq y'$. For (1), note that $h(g) \subset r$. It follows that $\sup(r \cup C) \leq \sup(h(g) \cup C)$, and hence $x \geq x'$. For (2), note that $r \subseteq h(g) \cup t(g)$. Because

$r \cup \{\neg c\} \subseteq h(g) \cup t(g) \cup \{\neg c\}$, we have:
$y \leq \sup(h(g) \cup t(g) \cup \{\neg c\})$
$\leq \sup(r \cup \{\neg c\}) = \sup(r) - \sup(r \cup C) = y'$. □

Theorem 5.2 is immediately applicable for computing $\text{uconf}(g)$ for a processed group g since the following itemsets needed to compute tight values for x and y are all within its candidate set: $h(g)$, $h(g) \cup C$, $h(g) \cup t(g)$, and $h(g) \cup t(g) \cup C$. There are $2^{|t(g)|} - 1$ rules derivable from a given group g, and the support of these four itemsets can be used to potentially eliminate them all from consideration. Note that if $h(g) \cup t(g) \cup C$ were frequent, then an algorithm such as Apriori would enumerate every derivable rule.

We have framed Theorem 5.2 in a manner in which it can be exploited even when the exact support information used above is not available. This is useful when we wish to prune a group before it is processed by using only previously gathered support information. For example, given an unprocessed group g, we cannot compute $\sup(h(g) \cup t(g) \cup \{\neg c\})$ to use for the value of y, but we can compute a lower-bound on the value. Given the parent node g_p of g, because $h(g_p) \cup t(g_p)$ is a superset of $h(g) \cup t(g)$, such a lower-bound is given by the observation below.

OBSERVATION 5.3: Given a group g and its parent g_p in the set-enumeration tree,
$\sup(h(g_p) \cup t(g_p) \cup \{\neg c\}) \leq \sup(h(g) \cup t(g) \cup \{\neg c\})$.

Conveniently, the support information required to apply this fact is immediately available from the candidate set of g_p.

In the following observation, we apply the support lower-bounding theorem from [4] to obtain another lower-bound on $\sup(h(g) \cup t(g) \cup \{\neg c\})$, again using only support information provided by the candidate set of g_p.

OBSERVATION 5.4: Given a group g and its parent g_p in the set-enumeration tree,

$$\sup(h(g) \cup \{\neg c\}) - \sum_{i \in t(g)} \sup(h(g_p) \cup \{\neg i, \neg c\}) \leq \sup(h(g) \cup t(g) \cup \{\neg c\}).$$

When attempting to prune an unprocessed group, Dense-Miner computes both lower-bounds and uses the greater of the two for y in theorem 5.2.

5.4 Bounding improvement

We propose two complementary methods to bound the improvement of any (frequent) rule derivable from a given group g. The first technique uses primarily the value of $\text{uconf}(g)$ described above, and the second directly establishes an upper-bound on improvement from its definition. Dense-Miner computes $\text{uimp}(g)$ by retaining the smaller of the two bounds provided by these techniques.

Bounding improvement using the confidence bound

The theorem below shows how to obtain an upper-bound on improvement by reusing the value of $\text{uconf}(g)$ along with another value z no greater than the confidence of the sub-rule of $h(g)$ with the greatest confidence.

THEOREM 5.5: The value of $\text{uconf}(g) - z$ where $z \leq \max(\forall r \subseteq h(g), \text{conf}(r))$ is an upper-bound on the improvement of any rule derivable from g.

Proof: Let r_s denote the sub-rule of $h(g)$ with the greatest confidence. Because r_s is a proper sub-rule of any rule r_d derivable from g, we know that $\text{conf}(r_d) - \text{conf}(r_s)$ is an upper-bound on $\text{imp}(r_d)$.
Because $\text{uconf}(g) \geq \text{conf}(r_d)$ and $z \leq \text{conf}(r_s)$, we have:
$\text{imp}(r_d) \leq \text{conf}(r_d) - \text{conf}(r_s)$
$\leq \text{conf}(r_d) - z$
$\leq \text{uconf}(g) - z$. □

Dense-Miner uses the previously described method for computing $\text{uconf}(g)$ when applying this result. Computing a tight value for z requires knowing the sub-rule r_s of $h(g)$ with the greatest confidence. Because r_s is not known, Dense-Miner instead sets z to the value of the following easily computed function:

$f_z(g) = \max(f_z(g_p), \text{conf}(h(g)))$ if g has a parent g_p,
$f_z(g) = \text{conf}(h(g))$ otherwise.

The fact that $f_z(g) \leq \max(\forall r \subseteq h(g), \text{conf}(r))$ follows from its definition. Its computation requires only the value of $f_z(g_p)$ where g_p is the parent of g, and the supports of $h(g)$ and $h(g) \cup C$ in order to compute $\text{conf}(h(g))$. The value can be computed whether or not the group has been processed because this information can be obtained from the parent group.

Bounding improvement directly

A complementary method for bounding the improvement of any frequent rule derivable from g is provided by the next theorem. This technique exploits strong dependencies between head items.

THEOREM 5.6: The following expression provides an upper-bound on the improvement of any frequent rule derivable from a given group g:

$$\frac{x}{x+y} - \frac{x}{x+y+\beta}$$

where x, y and β are non-negative integers such that
$y \leq \sup(h(g) \cup t(g) \cup \{\neg c\})$,
$\beta \geq \min(\forall i \in h(g), \sup((h(g) - \{i\}) \cup \{\neg c, \neg i\}))$, and
$x = \min(\max(\sqrt{y^2 + y\beta}, \text{minsup}), \sup(h(g) \cup C))$

Proof sketch: For any frequent rule r derivable from g, note that $\text{imp}(r)$ can be written as:

$$\frac{x'}{x'+y'} - \frac{x'+\alpha'}{x'+y'+\alpha'+\beta'}$$

where the first term represents $\text{conf}(r)$ (as in Theorem 5.2) and the subtractive term represents the confidence of the proper sub-rule of r with the greatest confidence. To prove the claim, we show how to transform this expression into the expression from the theorem statement, arguing that the value of the expression never decreases as a result of each transformation.

To begin, let the subtractive term of the expression denote the confidence of r_s, a proper sub-rule of r such that $r_s = r - \{i_m\}$ where i_m denotes the item i from $h(g)$ that minimizes $\sup((h(g) - \{i\}) \cup \{\neg c, \neg i\})$.

Since we can only decrease the value of the subtractive term by such a transformation, we have not decreased the value of the expression.

Now, given r and r_s, it is easy to show that $\alpha' \geq 0$, $y' \geq y$, and $\beta' \leq \beta$. Because the expression is anti-monotone in α' and y' and monotone in β', we can replace α' with 0, β' with β, and y' with y without decreasing its value.

We are now left with an expression identical to the expression in the theorem, except for x' occurring in place of x. Taking the derivative of this expression with respect to x' and solving for 0 reveals it is maximized when $x' = \sqrt{y^2 + y\beta}$. Note that for any rule derivable from g, x' must fall between $\sup(h(g) \cup C)$ and minsup. Given this restriction on x', the equation is maximized at $x' = \min(\max(\sqrt{y^2 + y\beta}, \text{minsup}), \sup(h(g) \cup C)) = x$. We can therefore replace x' with x without decreasing its value. The resulting expression, identical to that in the theorem statement, is thus an upper-bound on $\text{imp}(r)$. □

To apply this result to prune a processed group g, Dense-Miner sets y to $\sup(h(g) \cup t(g) \cup \{\neg c\})$ since the required supports are known. Computing a tight value for β ($\sup((h(g) - i_m) \cup \{\neg i_m, \neg c\})$ where i_m is the item in $h(g)$ that minimizes this support value) is not possible given the support values available in the candidate set of g and its ancestors. Dense-Miner therefore sets β to an upper-bound on $\sup((h(g) - i_m) \cup \{\neg i_m, \neg c\})$ as computed by the following function:

$f_\beta(g) = \min(f_\beta(g_p), \sup(h(g_p) \cup \{\neg i, \neg c\}))$ when g has a parent g_p and where i denotes the single item within the itemset $h(g) - h(g_p)$,

$f_\beta(g) = \infty$ otherwise.

This computation requires only the value of $f_\beta(g_p)$ which was previously computed by the parent, and the supports of candidate set members $h(g)$, $h(g) \cup C$, $h(g_p)$, and $h(g_p) \cup C$ in order to compute $\sup(h(g_p) \cup \{\neg i, \neg c\})$.

In applying theorem 5.6 to prune an unprocessed group g, Dense-Miner computes β as above. For y, it lacks the necessary support information to compute $\sup(h(g) \cup t(g) \cup \{\neg c\})$, so instead it computes a lower-bound on the value as described in section 5.3.

5.5 Bounding support

The value of $\text{usup}(g)$ is comparatively easy to compute because support is anti-monotone with respect to rule containment. For $\text{usup}(g)$, Dense-Miner simply uses the value of $\sup(h(g) \cup C)$. Other anti-monotone constraints, e.g. those discussed in [10], can be exploited with similar ease.

6. Item ordering

The motivation behind reordering tail items in the Generate-Next-Level function is to, in effect, force unpromising rules into the same portion of the search tree. The reason this strategy is critical is that in order for a group to be prunable, *every* sub-node of the group must represent a rule that fails to satisfy one of the constraints. An arbitrary ordering policy will result in a roughly even distribution of rules that satisfy the constraints throughout the search tree, yielding little pruning opportunities.

We experimented with several different ordering policies intended to tighten the bounds provided by the pruning functions. The strategy we found to work best exploits the fact that the computations for $\text{uconf}(g)$ and $\text{uimp}(g)$ both require a value $y \leq \sup(h(g) \cup t(g) \cup \{\neg c\})$, and the larger the value allowed for y, the tighter the resulting bound. The idea then is to reorder tail items so that many sub-nodes will have a large value for $\sup(h(g) \cup t(g) \cup \{\neg c\})$. This is achieved by positioning tail items which contribute to a large value of $\sup(h(g) \cup t(g) \cup \{\neg c\})$ last in the ordering, since tail items which appear deeper in the ordering will appear in more sub-nodes than those tail items appearing earlier. We have found that the tail items which contribute most to this value tend to be those with small values for $\sup(h(g) \cup \{\neg i, \neg c\})$. This can be seen from Observation 5.4 which yields a larger lower-bound on $\sup(h(g) \cup t(g) \cup \{\neg c\})$ when the value of $\sup(h(g) \cup \{\neg i, \neg c\})$ summed over every tail item is small. The policy used by Dense-Miner is therefore to arrange tail items in decreasing order of $\sup(h(g) \cup \{\neg i, \neg c\})$.

7. Post-processing

The fact that Dense-Miner finds all frequent, confident, large-improvement rules and places them into R follows from the completeness of a set-enumeration tree search and the correctness of our pruning rules, as established by the theorems from Section 5. Dense-Miner must still post-process R because it could contain some rules that do not have a large improvement.

Removing rules without a large improvement is non-trivial because improvement is defined in terms of all the proper sub-rules of a rule, and all such rules are not necessarily generated by the algorithm. A naive post-processor for removing rules without a large improvement might, for every mined rule, explicitly compute its improvement by generating and testing every proper sub-rule. Because Dense-Miner is capable of mining many long rules, such an approach would be too inefficient.

Instead, the post-processor first identifies some rules that do not have a large improvement by simply comparing them to the other rules in the mined rule set R. It compares each rule $r_1 \in R$ to every rule r_2 such that $r_2 \in R$ and $r_2 \subset r_1$. If ever it is found that $\text{conf}(r_1) - \text{conf}(r_2) < \text{minimp}$, then rule r_1 is removed because its improvement is not large. This step alone requires no database access, and removes almost all rules that do not have a large improvement.

To remove any remaining rules, the post-processor performs a set-enumeration tree search for rules that could potentially prove some rule in R does not have a large improvement. The main difference between this procedure and the mining phase is in the pruning strategies applied. For this search problem, a group g is prunable when none of its derivable rules can prove that some rule in R lacks a large improvement. This is determined by either of the following conditions:

- There exists no rule $r \in R$ for which $h(g) \subset r$;
- $\text{conf}(r) - \text{uconf}(g) \geq \text{minimp}$ for all rules $r \in R$ such that $h(g) \subset r$.

After groups are processed, any rule $r \in R$ is removed if there exists some group g such that $h(g) \subset r$ and $\text{conf}(r) - \text{conf}(h(g)) < \text{minimp}$. Because the search explores the set of all rules that could potentially prove some rule in R does not have a large improvement, all rules without a large improvement are identified and removed.

Our post-processor includes some useful yet simple extensions of the above for ranking and facilitating the understanding of rules mined by Dense-Miner as well as other algorithms. The improvement of a rule is useful as an interestingness and ranking measure to be presented to the user along with confidence and support. It is also often useful to present the proper sub-rule responsible for a rule's improvement value. Therefore, given an arbitrary set of rules, our post-processor determines the exact improvement of every rule, and associates with every rule its proper sub-rule with the greatest confidence (whether or not this sub-rule is in the original rule set). In rule-sets that are not guaranteed to have high-improvement rules (such as those extracted from a decision tree), the sub-rules can be used to potentially simplify, improve the generality of, and improve the predictive ability of the originals.

8. Evaluation

This section provides an evaluation of Dense-Miner using two real-world data-sets which were found to be particularly dense in [4].[1] The first data-set is compiled from PUMS census data obtained from http://augustus.csscr.washington.edu/census/comp_013.html. It consists of 49,046 transactions with 74 items per transaction, with each transaction representing the answers to a census questionnaire. These answers include the age, tax-filing status, marital status, income, sex, veteran status, and location of residence of the respondent. Similar data-sets are used in targeted marketing campaigns for identifying a population likely to respond to a particular promotion. Continuous attributes were discretized as described in [6], though no frequently occurring items were discarded. The second data-set is the connect-4 data-set from the Irvine machine learning database repository (http://www.ics.uci.edu/~mlearn/MLRepository.html). It consists of 67,557 transactions and 43 items per transaction. This data-set is interesting because of its size, density, and a minority consequent item ("tie games") that is accurately predicted only by rules with low support. All experiments presented here use the "unmarried partner" item as the consequent with the pums data-set, and the "tie games" item with the connect-4 data-set; we have found that using other consequents consistently yields qualitatively similar results.

Execution times are reported in seconds on an IBM IntelliStation M Pro running Windows NT with a 400 MHZ Intel Pentium II Processor and 128MB of SDRAM. Execution time includes runtime for both the mining and post-processing phases.

The minsup setting used in the experiments is specified as a value we call *minimum coverage*, where $\text{minimum coverage} = \text{minsup} / \text{sup}(C)$. In the context of consequent constrained association rule mining, minimum coverage is more intuitive than minimum support, since it specifies the smallest fraction of the population of interest that must be characterized by each mined rule.

8.1 Effects of minimum improvement

The first experiment (Figure 5) shows the effect of different minimp settings as minsup is varied. Minconf in these experiments is left unspecified, which disables pruning with the minimum confidence constraint. The graphs of the figure plot execution time and the number of rules returned for several algorithms at various settings of minimum support. Dense-miner is run with minimp settings of .0002, .002, and .02 (dense_0002, dense_002, and dense_02 respectively). We compare its performance to that of the Apriori algorithm optimized to exploit the consequent constraint (apriori_c). This algorithm materializes only those frequent itemsets that contain the consequent itemset.

The first row of graphs from the figure reveals that apriori_c is too slow on all but the greatest settings of minsup for both data-sets. In contrast, very modest settings of minimp allow Dense-Miner to efficiently mine rules at far lower supports, even without exploiting the minconf constraint. A natural question is whether mining at low supports is necessary. For these data-sets, the answer is yes simply because rules with confidence significantly higher than the consequent frequency do not arise unless minimum coverage is below 20%. This can be seen from Figure 7, which plots the confidence of the best rule meeting the minimum support constraint for any given setting.[2] This property is typical of data-sets from domains such as targeted marketing, where response rates tend to be low without focusing on a small but specific subset of the population.

The graphs in the second row of Figure 5 plot the number of rules satisfying the input constraints. Note that runtime correlates strongly with the number of rules returned for each algorithm. For apriori_c, the number of rules returned is the same as the number of frequent itemsets containing the consequent because there is no minconf constraint specified. Modest settings of minimp dramatically reduce the number of rules returned because most rules in these data-sets offer only insignificant (if any) predictive advantages over their proper sub-rules. This effect is particularly pronounced on the pums data-set, where a minimp setting of .0002 is too weak a constraint to keep the number of such rules from exploding as support is lowered. The increase in runtime and rule-set size as support is lowered is far more subdued given the larger (though still small) minimp settings.

[1] Both data-sets are available in the form used in these experiments from http://www.almaden.ibm.com/cs/quest.

[2] The data for this figure was generated by a version of Dense-Miner that prunes any group that cannot lead to a rule on the depicted support/confidence border. This constraint can be enforced during mining using the confidence and support bounding techniques from section 5.

FIGURE 5. Execution time and rules returned versus minimum coverage for the various algorithms.

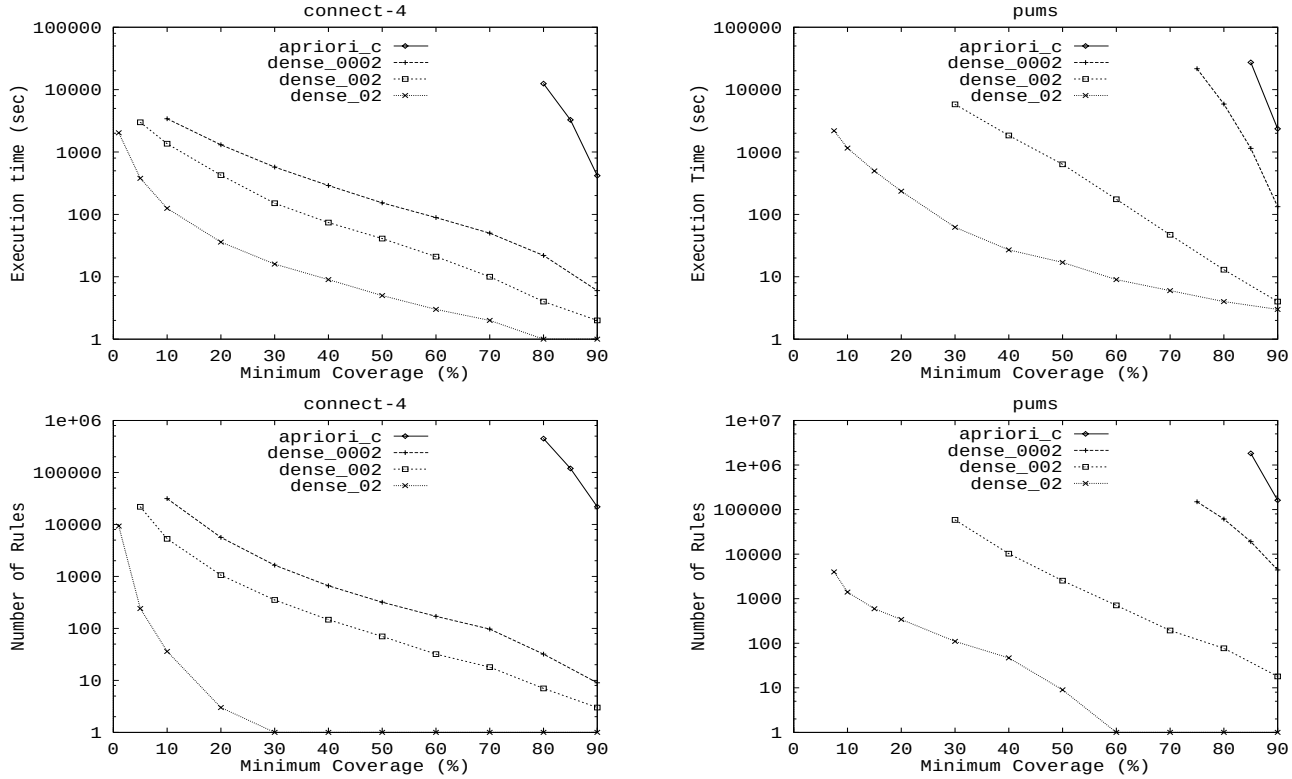


FIGURE 6. Execution time of dense_0002 as minconf is varied for both data-sets. Minimum coverage is fixed at 5% on pums and 1% on connect-4.

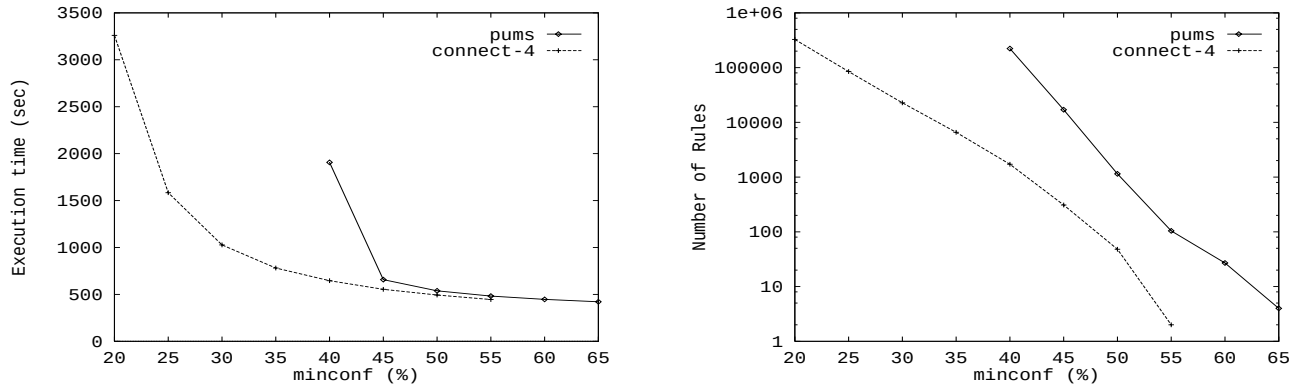
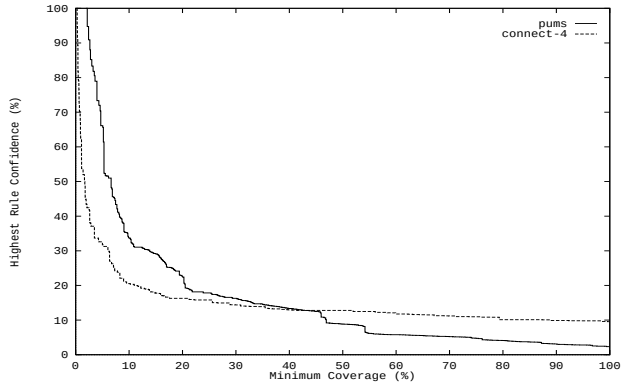


FIGURE 7. Maximum confidence rule mined from each data-set for a given level of minimum coverage.



8.2 Effects of minimum confidence

The next experiment (Figure 6) shows the effect of varying minconf while fixing minimp and minsup to very low values. With connect-4, we used a minimum coverage of 1%, and with pums, a minimum coverage of 5%. Minimp was set to .0002 with both data-sets. As can be extrapolated from the previous figures, the number of rules meeting these weak minimp and minsup constraints would be enormous. As a result, with these constraints alone, Dense-Miner exceeds the available memory of our machine.

The efficiency of Dense-Miner when minimum confidence is specified shows that it is effectively exploiting the confidence constraint to prune the set of rules explored. We were unable to use lower settings of minconf than those plotted because of the large number of rules. As minconf is increased beyond the point at which fewer than 100,000 rules are returned, the run-time of Dense-Miner rapidly falls to around 500 seconds on both data-sets.

8.3 Summary of experimental findings

These experiments demonstrate that Dense-Miner, in contrast to approaches based on finding frequent itemsets, achieves good performance on highly dense data even when the input constraints are set conservatively. Minsup can be set low (which is necessary to find high confidence rules), as can minimp and minconf (if it is set at all). This characteristic of our algorithm is important for the end-user who may not know how to set these parameters properly. Low default values can be automatically specified by the system so that all potentially useful rules are produced. Refinements of the default settings can then be made by the user to tailor this result. In general, the execution time required by Dense-Miner correlates strongly with the number of rules that satisfy all of the specified constraints.

9. Conclusions

We have shown how Dense-Miner exploits rule constraints to efficiently mine consequent-constrained rules from large and dense data-sets, even at low supports. Unlike previous approaches, Dense-Miner exploits constraints such as minimum confidence (or alternatively, minimum lift or conviction) and a new constraint called minimum improvement during the mining phase. The minimum improvement constraint prunes any rule that does not offer a significant predictive advantage over its proper sub-rules. This increases efficiency of the algorithm, but more importantly, it presents the user with a concise set of predictive rules that are easy to comprehend because every condition of each rule strongly contributes to its predictive ability.

The primary contribution of Dense-Miner with respect to its implementation is its search-space pruning strategy which consists of the three critical components: (1) functions that allow the algorithm to flexibly compute bounds on confidence, improvement, and support of any rule derivable from a given node in the search tree; (2) approaches for reusing support information gathered during previous database passes within these functions to allow pruning of nodes before they are processed; and (3) an item-ordering heuristic that ensures there are plenty of pruning opportunities. In principle, these ideas can be retargeted to exploit other constraints in place of or in addition to those already described.

We lastly described a rule post-processor that Dense-Miner uses to fully enforce the minimum improvement constraint. This post-processor is useful on its own for determining the improvement value of every rule in an arbitrary set of rules, as well as associating with each rule its proper sub-rule with the highest confidence. Improvement can then be used to rank the rules, and the sub-rules used to potentially simplify, generalize, and improve the predictive ability of the original rule set.

References

[1] Agrawal, R.; Imielinski, T.; and Swami, A. 1993. Mining Associations between Sets of Items in Massive Databases. In *Proc. of the 1993 ACM-SIGMOD Int'l Conf. on Management of Data*, 207-216.

[2] Agrawal, R.; Mannila, H.; Srikant, R.; Toivonen, H.; and Verkamo, A. I. 1996. Fast Discovery of Association Rules. In *Advances in Knowledge Discovery and Data Mining*, AAAI Press, 307-328.

[3] Ali, K.; Manganaris, S.; and Srikant, R. 1997. Partial Classification using Association Rules. In *Proc. of the 3rd Int'l Conference on Knowledge Discovery in Databases and Data Mining*, 115-118.

[4] Bayardo, R. J. 1998. Efficiently Mining Long Patterns from Databases. In *Proc. of the 1998 ACM-SIGMOD Int'l Conf. on Management of Data*, 85-93.

[5] Berry, Michael J. A. and Linoff G. S. 1997. *Data Mining Techniques for Marketing, Sales and Customer Support*, John Wiley & Sons, Inc.

[6] Brin, S.; Motwani, R.; Ullman, J.; and Tsur, S. 1997. Dynamic Itemset Counting and Implication Rules for Market Basket Data. In *Proc. of the 1997 ACM-SIGMOD Int'l Conf. on the Management of Data*, 255-264.

[7] Cohen, W. W. 1995. Fast Effective Rule Induction. In *Proc. of the 12th Int'l Conf. on Machine Learning*, 115-123.

[8] International Business Machines, 1996. *IBM Intelligent Miner User's Guide*, Version 1, Release 1.

[9] Klemettinen, M.; Mannila, P.; Ronkainen, P.; and Verkamo, A. I. 1994. Finding Interesting Rules from Large Sets of Discovered Association Rules. In *Proc. of the Third Int'l Conf. on Information and Knowledge Management*, 401-407.

[10] Ng, R. T.; Lakshmanan, V. S.; Han, J.; and Pang, A. 1998. Exploratory Mining and Pruning Optimizations of Constrained Association Rules. In *Proc of the 1998 ACM-SIGMOD Int'l Conf. on the Management of Data*, 13-24.

[11] Rymon, R. 1992. Search through Systematic Set Enumeration. In *Proc. of Third Int'l Conf. on Principles of Knowledge Representation and Reasoning*, 539-550.

[12] Shafer, J.; Agrawal, R.; and Mehta, M. 1996. SPRINT: A Scalable Parallel Classifier for Data-Mining. In *Proc. of the 22nd Conf. on Very Large Data-Bases*, 544-555.

[13] Smythe, P. and Goodman, R. M. 1992. An Information Theoretic Approach to Rule Induction from Databases. *IEEE Transactions on Knowledge and Data Engineering*, 4(4):301-316.

[14] Srikant, R.; Vu, Q.; and Agrawal, R. 1997. Mining Association Rules with Item Constraints. In *Proc. of the Third Int'l Conf. on Knowledge Discovery in Databases and Data Mining*, 67-73.

[15] Webb, G. I. 1995. OPUS: An Efficient Admissible Algorithm for Unordered Search. In *Journal of Artificial Intelligence Research*, 3:431-465.