A New and Versatile Method for Association Generation

Amihood Amir* Ronen Feldman† Reuven Kashi‡
Georgia Tech and Bar-Ilan University
Bar-Ilan University
Bar-Ilan University

Abstract

Current algorithms for finding associations among the attributes describing data in a database have a number of shortcomings:

1. Their performance time grows dramatically as the minimum *support* is reduced. Consequently, applications that require associations with very small support have prohibitively large running times.
2. They assume a *static* database. Some applications require generating associations in real-time from a dynamic database, where transactions are constantly being added and deleted. There are no existing algorithms to accommodate such applications.
3. They can only find associations of the type where a conjunction of attributes implies a conjunction of different attributes. It turns out that there are many cases where a conjunction of attributes implies another conjunction only in case certain other attributes are *excluded*. To our knowledge, there is no current algorithm that can generate such *excluding associations*.

We present a novel method for association generation, that answers all three above desiderata. Our method is inherently different from all existing algorithms, and especially suitable to textual databases with binary attributes. At the heart of our algorithm lies the use of subword trees for quick indexing into the required database statistics. We tested our algorithm on the Reuters-22173 database with satisfactory results.

*Department of Mathematics and Computer Science, Bar-Ilan University, 52900 Ramat-Gan, Israel, (972-3)531-8770; amir@cs.biu.ac.il; Partially supported by NSF grant CCR-95-31939 and the Israel Ministry of Science and the Arts grant 6297.

†Department of Mathematics and Computer Science, Bar-Ilan University, 52900 Ramat-Gan, Israel, (972-3)531-8629; feldman@cs.biu.ac.il; Partially supported by the Israel Ministry of Science and the Arts grant 8615.

‡Department of Mathematics and Computer Science, Bar-Ilan University, 52900 Ramat-Gan, Israel, (972-3)531-7529; kashi@cs.biu.ac.il.

1 Introduction

Data Mining has recently been recognized as a promising new field in the intersection of databases, artificial intelligence, and machine learning [12, 5]. While traditional database query tools allow a user to retrieve records based on the content of each record in isolation, the Knowledge Discovery in Databases (KDD) paradigm provides tools for acquiring information based on patterns appearing *across* records.

In a ground breaking paper, Agrawal, Imielinski and Swami [2] introduced the concept of *association rules* and gave an algorithm for finding such rules. An association rule is an expression of the form $S_1 \Rightarrow S_2$ where S_1, S_2 are sets of attributes with sufficient *support* and *confidence*. Support and confidence are defined as follows.

1. There is enough *support* (*minsupp*) for the rule $S_1 \Rightarrow S_2$ if the number of records whose attributes include $S_1 \cup S_2$ is at least *minsupp*. (Note that in the literature the support is usually measured as a given *fraction* of the database. However, we deal with extremely small supports and thus choose to define the support absolutely.)

2. We have enough *confidence* ($minconf$) in the association rule if the ratio of records having attributes that include $S_1 \cup S_2$ over records having attributes that include S_1 is at least $minconf$.

Algorithms for finding association rules appear in [1, 11, 8, 9, 13]. These algorithms find *covers*, i.e. sets of attributes with large enough support. They then check the confidence on suitable partitions of the covers. In the worst case these algorithms have exponential time complexity. In reality, though, their running time on the tested databases is manageable. The reasons for this are the heuristics they implement. The heuristics are all based on the support. The speed of the algorithm is proportional to the size of the minimum support. The higher the support – the faster the algorithm. For very small supports these algorithms break down.

Consider the following two application domains that require mining for associations. The first is a scenario that involves finding associations between labels of articles in textual collections [5, 6, 7]. Another important example is in a medical database where transactions are medical records.

Both of these important applications have needs that are not answered by the existing algorithms.

1. **Small Support:** Medicine is replete with rare but lethal cases. Under these circumstances it is better to err on the side of safety and consider associations with small support. What is needed, then, is a completely new method of generating associations in a *support independent* manner.

   Current association generation algorithm do not efficiently handle such small supports, often in the single digit range.

2. **Dynamic Databases:** In a fast-changing database (growing or shrinking), the user may need results in real time as new records accumulate or change and the association needs to reflect all known data.

In [4] two methods for real time generation of associations in incremental databases were developed. They were tested and compared on the Reuters-22173 database using the KDT data mining system. While the methods of [4] performed quite well on the Reuters-22173 test database, they suffer from two main drawbacks. (1) They are both support dependent, and (2) They assume only an *increasing* database, but can not handle a *decreasing* database.

3. **General Associations:** In our definition of association we followed Agrawal, Imielinski and Swami [1]. Their definition of an association rule means that a conjunction of attributes implies a conjunction of other attributes, $A_1 \wedge A_2 \wedge \cdots \wedge A_i \Rightarrow B_1 \wedge \cdots \wedge B_j$. Recently, Mannila and Toivonen [10] discuss the theoretical option of having a general boolean formula implying another general boolean formula. Many such formulae are meaningless from a data mining perspective. For example, $A_1 \wedge A_2 \wedge \cdots \wedge A_i \Rightarrow \neg B_1 \wedge \cdots \wedge \neg B_j$ for all subsets $\{B_1, \ldots, B_j\}$ where it is not the case that $A_1 \wedge A_2 \wedge \cdots \wedge A_i \Rightarrow B_\ell, \quad \ell = 1, \ldots, j$. It is clearly not worthwhile to seek such meaningless associations, especially considering the fact that their number is exponential in the number of attributes (587 in the Reuters-22173, for example).

   However, a possibly interesting case is associations of the following form: $A_1 \wedge A_2 \wedge \cdots \wedge A_i \Rightarrow B_1 \wedge \cdots \wedge B_j$ where some of the A_ℓ's are negations. This means that there is sufficient support and confidence for a rule, only provided that certain attributes are *not present.*

   In structured databases it is quite often important to consider attributes that are *missing* (i.e. marriage (yes/no), damage (yes/no), etc.). However, there are domains where excluding information is interesting in a sparse database as well (the type suited for our algorithms). Consider a medical example where certain conditions lead to a conclusion provided some action is *not taken* (e.g. eating proper foods and not smoking leads to long life. It may be the case that just eating proper food may not mean a long life).

   Example: Consider a case where there is sufficient support for set $\{A, B, D\}$ but not enough confidence for $A \wedge B \Rightarrow D$. On the other hand, if we *exclude* C, then there is both sufficient support and confidence for the rule. We can conclude that "A and B imply D when C does not occur", i.e. $A \wedge B \wedge \neg C \Rightarrow D$.

   Such *excluding associations* are meaningful and important, yet the current algorithms are incapable of producing them.

In this paper we present a novel idea for generating associations. Our method uses subword tree techniques to mine for associations and is fundamentally different from all previously known algorithms. Our algorithm can handle very small supports, handles dynamic databases, and finds associations with a single exclusion.

Our algorithm's worst case time is exponential in the maximum cover size, as opposed to an exponent in the number of attributes or record size. In practice the number of attributes in a cover turns out to be rather small (7 in the Reuters-22173 database with support 10, which is equal to 0.045%), while the number of attributes or the record size may be much larger (587 and 26, resp. in the Reuters-22173 database).

In addition, our algorithm reads the database only once. Since the database is normally on external memory, the main time devoted to the algorithm is the I/O time. Consequently, the fact that our algorithm reads the data only once significantly reduces the running time. As a result, the small exponential multiple on the number of machine instructions is not critical.

Our method is especially suitable for textual databases, where there is a very large number of attributes, but the length of an individual transaction is small. Under such circumstances it scales reasonably well. For example, if the maximum cover size is 5, then the generated auxiliary databases will be roughly 30 times larger than the initial database. In addition, we only handle binary attributes, where for every transaction either an attribute exists or it does not.

2 Problem Definition

The following is a formal statement of the problem based on the description of the problem in [2, 3, 11]. Let $I = \{i_1, i_2, ..., i_m\}$ be a set of *attributes*, also called *items*. Let D be a set of variable length transactions over I. Each transaction contains a set of items $\{i_i, i_j, ..., i_k\} \subset I$. A set of items is called an *itemset*. The number of items in an itemset is the *length* (or the *size*) of an itemset. An itemset of length k is referred to as a $k - itemset$.

An *association rule* is a relation of the form $S_1 \Rightarrow S_2$, where $S_1, S_2 \subset I$, and $S_1 \cap S_2 = \emptyset$. S_1 is called the *antecedent* of the rule, and S_2 is called the *consequent* of the rule. We follow the literature [2, 3, 11] in denoting associations as a relation between sets of attributes. A generalization, explored by Mannila and Toivonen [10], is denoting associations as a relation between boolean formulae. Thus, a rule of the form $\{A_1, A_2, \ldots, A_n\} \Rightarrow \{B_1, B_2, \ldots, B_m\}$ will be denoted in [10] as $A_1 \wedge A_2 \wedge \cdots \wedge A_n \Rightarrow B_1 \wedge B_2 \wedge \cdots \wedge B_m$. The formal definition of an association rule appears below.

Each rule has an associated measure called *support*. For an itemset $S \subset I$, the *support* of S is the number of transactions in D that contain the itemset S. (Note that in the literature the support is usually measured as a given *fraction* of the database. However, we deal with extremely small supports and thus choose to define the support absolutely). We denote by $supp(S)$ the *support* of S. The support of the rule $S_1 \Rightarrow S_2$ is defined as $supp(S_1 \cup S_2)$. An itemset that has support greater than or equal to a specified minimum support threshold is called a *covering itemset* or shortly *cover*.

A rule $S_1 \Rightarrow S_2$ has a measure of its strength called $confidence$ defined as the ratio of number of transactions in D that contain itemset $S_1 \cup S_2$ over number of transactions that contain itemset S_1.

The problem of mining association rules is to generate all rules $S_1 \Rightarrow S_2$ that have both support and confidence greater than or equal to some user specified minimum support (*minsupp*) and minimum confidence ($minconf$) thresholds respectively, i.e. for regular associations:

$$supp(S_1 \cup S_2) \geq minsupp$$

and

$$\frac{supp(S_1 \cup S_2)}{supp(S_1)} \geq minconf.$$

Example: Let support threshold $minsupp = 2$ and confidence threshold $minconf = 0.75$, and let the database D consist of the following transactions:

$$\{1,2\},\ \{3,4,5\},\ \{2,3,8\},\ \{1,5,8\},\ \{2,3\},\ \{2,3,8\}.$$

The *covers*, i.e. sets with *support* $\geq$ *minsupp*, and their corresponding *supports* are plotted in the table below.

Cover	{1}	{2}	{3}	{5}	{8}	{2,3}	{2,8}	{3,8}	{2,3,8}
Support	2	4	4	2	3	3	2	2	2

All itemsets that have support *less* than minsupp are not covers. For instance, itemset {1,5,8} has support = 1 which is less than minsupp = 2. The database satisfies $\{3,8\} \Rightarrow \{2\}$, $\{2,8\} \Rightarrow \{3\}$, $\{3\} \Rightarrow \{2\}$ and $\{2\} \Rightarrow \{3\}$, as {2,3,8} is a cover. The confidence of these rules are $\frac{2}{2} = 1$, $\frac{2}{2} = 1$, $\frac{3}{4} = 0.75$ and $\frac{3}{4} = 0.75$ correspondingly, which are greater than or equal to $minconf = 0.75$.

The problem of finding association rules can be decomposed into the following two subproblems:

1. All covers, i.e. itemsets that have the minimum support, are generated.
2. All the rules that have minimum confidence are generated in the following naive way: For every covering itemset S and any $S_2 \subset S$, let $S_1 = S - S_2$. If the rule $S_1 \Rightarrow S_2$ has the minimum confidence, then it is a valid rule.

If all covers and their supports are given, one can generate rules in a brute-force manner, by considering all partitions of the powerset of every cover. Thus, the existing literature concerned itself mainly with generating all covering itemsets and their supports. In section 3 we present an efficient algorithm for generating all covers. In addition, we show an efficient and fast method for generating all associations.

Properties: Coverage is monotone with respect to contraction of the set: if S is a covering and $S' \subseteq S$ then S' is also covering. On the other hand, association rules do not have monotonicity properties with respect to expansion or contraction of the *antecedent* (left hand side of the rule): if $S \Rightarrow S'$ holds, then $S \cup S'' \Rightarrow S'$ does not necessarily hold, and if $S \cup S'' \Rightarrow S'$ holds, then $S \Rightarrow S'$ does not necessarily hold. In the first case the rule $S \cup S'' \Rightarrow S'$ does not necessarily have enough support or confidence, and in the second case the rule $S \Rightarrow S'$ does not necessarily have enough confidence.

3 A New Approach

We use the *trie* data structure to generate covers. Using the trie enables generating covers efficiently even for very small supports. In addition, the information stored in the trie can be used later for exceedingly fast queries for new types of associations.

3.1 The Trie Data Structure

A *trie* ([14]) is a data structure that allows a set of strings to be stored and updated, and allows membership queries. We formally define a trie.

Definition 3.1 *Let* $D = \{S_1, \ldots, S_k\}$ *be a set of strings over alphabet* Σ*,* $S_i = S_i[1] \cdots S_i[n_i]$*,* $i = 1, \ldots, k$*, and let* $\$ \notin \Sigma$ *be a special character. A trie* T_D *of* D *is a labeled rooted tree where the root represents the null string* λ *and every edge is labeled with a character from* $\Sigma \cup \$$*, and each string* $S_i \in D$ *is represented by a leaf* l_i *such that the concatenation of the edge labels from the root to* l_i *is the string* $S_i\$$*. It is defined recursively as follows:*

> *Outgoing edges from depth-0 node (root): Let* $L_0 = \cup_{i=1}^{k}\{S_i[1]\}$*. Then the root has exactly* $|L_0|$ *children. The edge to each child is labeled by a different element of* L_0 *(thus no two outgoing edges have the same label).*
>
> *Outgoing edges from depth-d node* x*: Let* x *be a depth-d node such that the concatenation of labels on the path from the root to* x *is* $a_1 \cdots a_d$*. Let* $S_{i_1}, \ldots, S_{i_m}$ *be the strings in* C *whose d-length prefixes equal* $a_1 \cdots a_d$*, i.e.* $S_{i_l}[j] = a_j$*,* $l = 1, \ldots, m$*;* $j = 1, \ldots, d$*. Let* $L_d^x = \cup_{l=1}^{m}\{S_{i_l}[d+1]\}$*. Then* x *has exactly* $|L_d^x|$ *children. The edge to each child is labeled by a different element of* L_d^x*.*

Constructing and updating a trie is straightforward (see e.g. [14]).

3.2 Using the Trie

Our idea is to *preprocess* the database and construct a data structure that encodes the information needed for association generation. Once that data structure is constructed there is really no further need to access the original database. The new data structure can then be used to generate various different types of associations. We also show that updating the new data structure when the database is changed is a fast and simple process.

In the preprocessing phase we construct a trie of the database. Since the database entries are *sets* of attributes and a trie is defined on *strings* some conversion is necessary. The first step is numbering the different attributes. We then consider every set as a string sorted by order of the attribute number.

We would like every node on the trie to be a potential cover. Unfortunately, every subset of a record is a potential cover. It would seem that an exponential space is required, which would make the idea prohibitively expensive for any but a database with very small records. However, the situation turns out to be a lot better than that. Because cover lengths are usually not large (at most 7, in the Reuters-22173 even for the very small support of 10) we can set a relatively small value k as the maximum cover size and only consider attribute subsets of size up to k for inclusion in the trie. Thus, for a record of m attributes, where k is the value taken as the bound of the largest cover, we consider only the $\sum_{i=1}^{k} \binom{m}{i}$ different i-element subsets of the m attributes, for $i = 1, \ldots, k$.

We can now give an overview of the preprocessing phase:

Preprocessing:

1. Scan the database. For every record:
   (a) For $i = 1, \ldots, k$, consider every different i-element subset of attributes as a string of the respective attribute numbers (sorted) and add it to the trie.
   (b) For every added string, increment the counter of the number of strings ending in its node.

Note that this implementation is inherently fully dynamic since addition and, similarly, deletion of strings can be done whenever a record is added or deleted. For a pseudocode of the trie construction see Figure 1.

Example: Let the database D consist of the following (sorted) transactions:

$$T_1 = \{1, 2\}, T_2 = \{1, 3, 4, 5\}, T_3 = \{2, 3, 4\}, T_4 = \{2, 3, 4, 5\}, T_5 = \{2, 3, 4\}.$$

Let k, the maximum cover size, be the maximum transaction length. Therefore, for each transaction the algorithm inserts all its subsets into the trie, and increments by one the counter of the appropriate node of each subset.

The resulting trie from the above example database is visually shown in Figure 2. Each node is associated with two parameters. The first, in curly brackets, is the itemset represented by this node, and second, denoted by #, is the support of this itemset. In addition, the last boldfaced item in each itemset of each node represents the label of the edge pointing to this node. Thus, the concatenation of the edge labels from the root to a node is represented by an itemset in this node.

For the rest of this paper we refer in all our examples to the database and the trie presented in this example.

Conclusion of Preprocessing Phase: At the end of the preprocessing phase we have a data structure encoding all potential covers and the number of transactions that include those sets

```
build_trie(input: database D)
λ ← root of the trie T_D
for each unread transaction T ∈ D do
    read in transaction T
    sort T in lexicographic order
    for i = 1 to k do
        for each i-itemset S ⊆ T do
            let S = S[1]···S[i]
            current_node ← λ
            for d ← 1 to i do
                if there exist an edge (current_node, n_d) labeled with S[d] then
                    current_node ← n_d
                    if d = i then
                        current_node.counter++
                else
                    create a path < current_node, n_d, n_{d+1}, ..., n_i >
                    with edge labels S[d], S[d+1], ..., S[i] respectively
                    n_i.counter ← 1
                    exit inner for loop
            end
        end
    end
end
```

Figure 1: *Algorithm build_trie for constructing the trie*

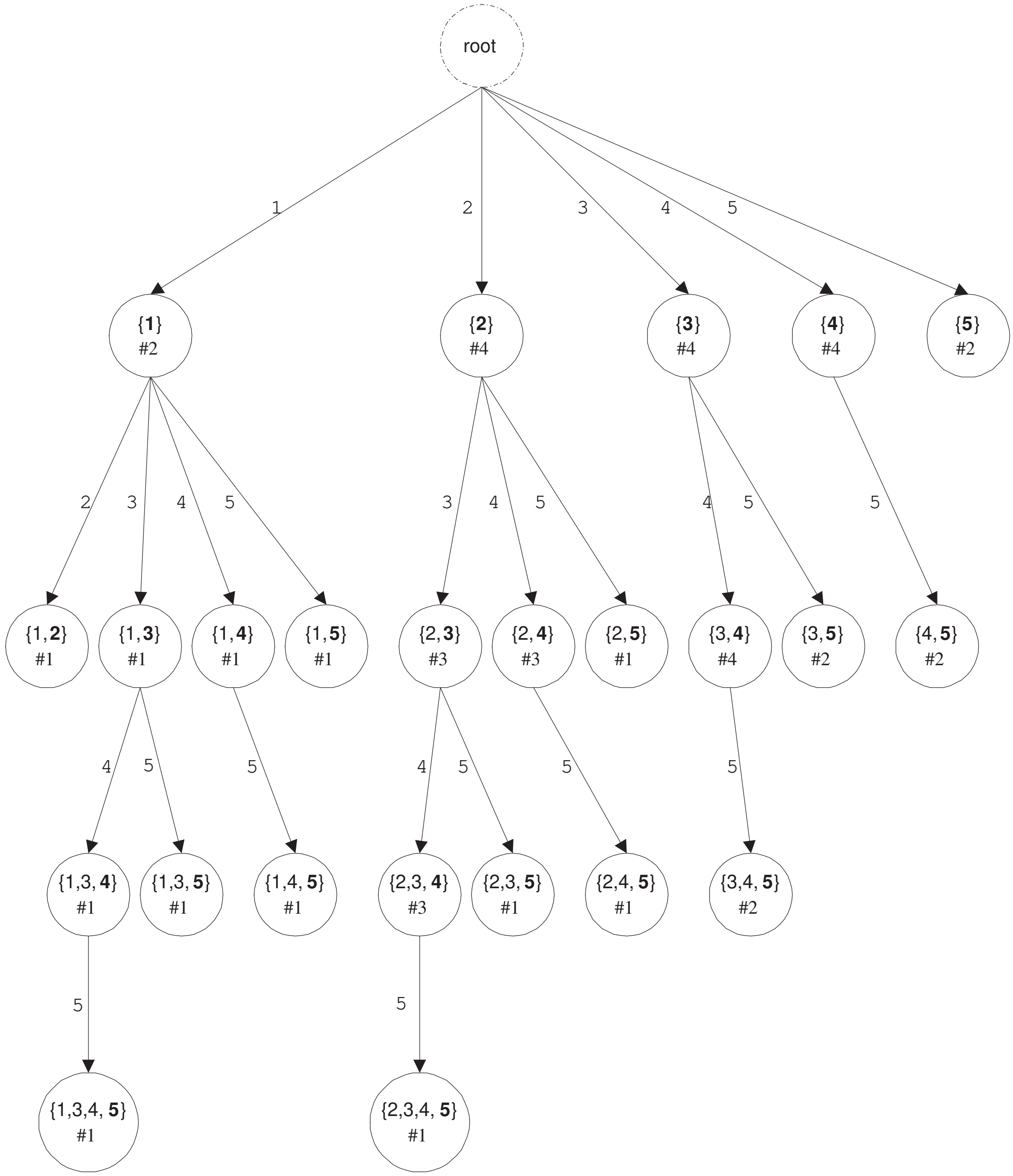


Figure 2: *The Resulting Trie From The Example Database*

among their attributes, with an extremely fast and efficient access method to every cover. The data structure is dynamically updated. Its size is quite manageable (see experimental results) and it encodes data that allows instantaneous recognition of covers, efficient support-independent generation of associations, as well as generation of excluding associations.

4 Association Generation

All current algorithms generate association rules by the exponential-time naive method of testing the confidence of all subsets of the cover. The following lemma provides an idea for more efficient generation of rules from covers.

Lemma: Let S, S_1, S_2 be *mutually disjoint* sets of attributes.
If $S \Rightarrow S_1 \cup S_2$ is an association rule, *then* also $S \cup S_1 \Rightarrow S_2$ and $S \cup S_2 \Rightarrow S_1$ are association rules.
Proof: Easy by *Venn Diagrams*.

The meaning of the lemma is that once all associations of the form $S \Rightarrow \{a\}$ have been generated, where a is a single attribute, we can narrow down the space of potential attributes of the form $S \Rightarrow \{a, b\}$. In particular, only if *both* associations $S \cup \{a\} \Rightarrow \{b\}$ and $S \cup \{b\} \Rightarrow \{a\}$ exist, is there any chance for $S \Rightarrow \{a, b\}$ to exist. A similar argument holds for larger sets in the right hand side of the association.

The algorithm in Figure 3 generates all associations.

Example: Using the database of the previous example, whose trie appears in Figure 2, let $minsupp = 2$ and let $minconf = 0.8$.

The algorithm traverses the trie and visits each node with support greater than 2. For each such node the algorithm outputs the itemset that is represented by this node, as a *cover*. Each subtree rooted at a node with support less than 2 is ignored.

As each cover is generated the algorithm generates the rules from this cover. For instance, the itemset $\{3, 4, 5\}$ is a cover with support equal to 2. First, all rules with $1 - consequent$ are generated. These are: $\{4, 5\} \Rightarrow \{3\}$ and $\{3, 5\} \Rightarrow \{4\}$, with confidence 1. The rule $\{3, 4\} \Rightarrow \{5\}$ is not valid since its confidence is 0.5, which is less than $minconf = 0.8$. Consequently, the only rule with $2 - consequent$ that can be generated from the cover $\{3, 4, 5\}$ is $\{5\} \Rightarrow \{3, 4\}$. Indeed, the rule $\{5\} \Rightarrow \{3, 4\}$ is valid since its confidence is equal to 1.

Association Generation Algorithm:

1. build the $trie$ from the transaction set D.
2. for each node with enough $support$ do:
   (a) Let $S = \{S_1, \ldots, S_k\}$ be the concatenation of the edge labels on the path from the $root$ to that $node$. (Each S_i represents an attribute)
   (b) For each S_i, $i = 1, \ldots, k$, extract S_i from S and retrieve its $support$ from the trie (by following the path without S_i).
   (c) If $\frac{supp(S)}{supp(S-S_i)} \geq minconf$, then $S - S_i \Rightarrow S_i$ is an $association\ rule$.

   (At the end of that stage we have all rules $S - S_i \Rightarrow S_i$ where $|S_i| = 1$. We now generate multiple rules by taking combination of these rules as in the lemma above.)
3. for each cover that generated association rules do:
   (a) Take pairs of rules generated in the previous step, i.e. the consequent has size 1, and generate new rules with consequents of size 2.
   (b) For each such new potential rule check its $confidence$ as in phase two.
4. In a similar fashion, for $k = 3, \ldots, m$, where m is the number of rules $S - S_i \Rightarrow S_i$ such that S_i is a single attribute, continue to generate new rules with consequents of size k.

End.

Figure 3: *Association Generation Algorithm*

Note that according to the lemma above the algorithm does not check the confidence on all partitions of the cover $\{3,4,5\}$. Therefore, it does not need to check the rules $\{3\} \Rightarrow \{4,5\}$ and $\{4\} \Rightarrow \{3,5\}$ which are known in advance to be not valid, since the rule $\{3,4\} \Rightarrow \{5\}$ does not hold.

5 Excluding Associations

We previously defined the meaning of $supp(S)$ where S is a set of attributes and $supp(S_1 \Rightarrow S_2)$ where $S_1 \Rightarrow S_2$ is a *regular association.*

Let S be a set of attributes $\{A_1, \ldots, A_n\}$. Denote the set $\{\neg A_1, \ldots, \neg A_n\}$ by $\neg S$.

Excluding associations are of the form $S_1 \cup \neg S_2 \Rightarrow S_3$, where S_1, S_2, S_3 are mutually disjoint. Recall that the intuitive meaning of $S_1 \cup \neg S_2 \Rightarrow S_3$ is that the attributes in S_1 imply the attributes in S_3 **provided** that the attributes in S_2 **do not** appear in the transaction.

In the boolean formula notation such a rule will be written as $A_1 \wedge A_2 \wedge \cdots \wedge A_n \wedge \neg B_1 \wedge \neg B_2 \wedge \cdots \wedge \neg B_m \Rightarrow C_1 \wedge C_2 \wedge \cdots \wedge C_\ell$, where $S_1 = \{A_1, \ldots, A_n\}$, $\neg S_2 = \{\neg B_1, \ldots, \neg B_m\}$ and $S_3 = \{C_1, \ldots, C_\ell\}$.

Definition: We say that $S_1 \cup \neg S_2 \Rightarrow S_3$ is an *excluding association* if S_1, S_2, S_3 are mutually disjoint sets of attributes and the following conditions hold:

$$(E_1)\ \frac{supp(S_1 \cup S_3)}{supp(S_1)} < minconf$$

$$(E_2)\ supp(S_1 \cup S_3) - supp(S_1 \cup S_2 \cup S_3) \geq minsupp, \text{ and}$$

$$(E_3)\ \frac{supp(S_1 \cup S_3) - supp(S_1 \cup S_2 \cup S_3)}{supp(S_1) - supp(S_1 \cup S_2)} \geq minconf.$$

In the rest of this paper we will only consider excluding associations where S_2 and S_3 have a single attribute. It should be noted that our algorithm can be generalized to larger sets by considering all subsets of a cover. We did not implement this direction because of time complexity considerations.

An excluding association can be located by a depth first search (DFS) on the trie, in a similar manner to the DFS performed for finding associations. For every node with enough support, let the path to the node be S_1. We need to pick out the pairs $\{a\}$ and $\{b\}$ as S_2 and S_3 for which conditions (E_1), (E_2) and (E_3) hold. Clearly, trying all attributes is prohibitively expensive. Instead, we only check conditions (E_1), (E_2) and (E_3) for pairs S_2 and S_3 where both $supp(S_1 \cup S_3)$ and $supp(S_1 \cup S_2)$ are non-zero, i.e. appear in the trie. The reason is that if $supp(S_1 \cup S_3) = 0$ then condition (E_2) can not hold, and if $supp(S_1 \cup S_2) = 0$ then conditions (E_1) and (E_3) can not both hold.

Searching through the trie for all such S_2, S_3 is also too time consuming. Rather, we achieve this as follows. Consider a node in the trie where S_1 is the set of the attributes on the path from the

Algorithm for Creating Lists of Pointers

Do DFS on the trie.

For each node N do

Let $\{s_1, \ldots, s_k\}$ be the set of edge labels on the path to N.

For $i = 1$ to k do

Add to the list of pointers of the trie node M whose path is $\{s_1, \ldots, s_k\} - \{s_i\}$ a pointer to N, if M has sufficient support.

end

end

end Algorithm

Figure 4: *Algorithm for creating list of pointers to* $S_1 \cup \{a\}$

root to the node. If $supp(S_1) > minsupp$ then we attach to the node a list of pointers to all trie nodes whose path is $S_1 \cup a$ where a is an attribute not in S_1. Using such a list makes checking conditions (E_1), (E_2) and (E_3) a simple matter.

We create such a list by a DFS on the trie. For every node, and every symbol σ on the path S of that node, update the list of $S - \{\sigma\}$ to point to the node of S. The time for constructing these lists is, therefore, proportional to the sum of the path lengths in the entire trie.

The algorithm in Figure 4 creates the list of pointers to $S_1 \cup \{a\}$.

The Excluding Association Generation Algorithm outline is the same as the Association Generation Algorithm, with the following difference. Step 2. should look as follows:

2. for each node whose support is no less than $minsupp$ do:
   (a) Let $S_1 = \{s_1, \ldots, s_k\}$ be the set of edge labels on the path from the *root* to that *node*. (Each s_i represents an attribute.)
   (b) Let $A = \{a_1, \ldots, a_\ell\}$ be the list of attributes such that $supp(S_1 \cup a_i) \neq 0, \quad i = 1, \ldots, \ell$. For each pair $a, b \in A$, let $S_2 = \{a\}$ and S_3 $\{b\}$. Check conditions $(E_1), (E_2)$, and (E_3). If they all hold then $S_1 \cup \neg S_2 \Rightarrow S_3$.

Example: Consider again the example database in Figure 2, this time where the minimum support $minsupp$ and the minimum confidence $minconf$ are 2 and 0.8 respectively.

Consider the cover itemset $\{2, 3, 4\}$ which has support 3. The *regular* rules $\{2, 4\} \Rightarrow \{3\}$ and $\{2, 3\} \Rightarrow \{4\}$ hold. But the regular rule $\{3, 4\} \Rightarrow \{2\}$ does not hold since its confidence equals

$\frac{3}{4} = 0.75$, which is less than $minconf$ that equals 0.8.

On the other hand, if we *exclude* 5 then we get a new *excluding rule* $\{3, 4, \neg 5\} \Rightarrow \{2\}$ that holds. It has both sufficient support and confidence as follows.

Its support is:

$$supp(\{2,3,4\}) - supp(\{2,3,4,5\}) = 3 - 1 = 2$$

and its confidence is:

$$\frac{supp(\{2,3,4\}) - supp(\{2,3,4,5\})}{supp(\{3,4\}) - supp(\{3,4,5\})} = \frac{3-1}{4-2} = \frac{2}{2} = 1.$$

6 Experimental Results

We ran our tests on the Reuters-22173 database. The Reuters-22173 text categorization test collection is a set of documents that appeared on the Reuters newswire in 1987. The 22173 documents were assembled and indexed with categories by personnel from Reuters Ltd. and Carnegie Group, Inc. in 1987. Further formatting and data file production was done in 1991 and 1992 by David D. Lewis and Peter Shoemaker. The documents were tagged with 587 attributes. We treat each document as a single transaction, where the attributes are the keywords with which the document is tagged. The size of the Reuters-22173 textual database is 25mb. The size of the Reuters-22173 ASCII attribute set is 1mb.

The platform we used was an UltraSPARC 1000 with 64mb main memory.

6.1 Trie Construction

We constructed three tries. One was a full trie with all subsets of every transaction. The other tries had maximum cover sizes 10 and 7. The following table summarizes the time and space it took to construct each of the tries. In the table below, the first row is the size of the trie, the second row is the time it took to construct the trie, and the third row is the construction time and saving the trie on the disk for future use.

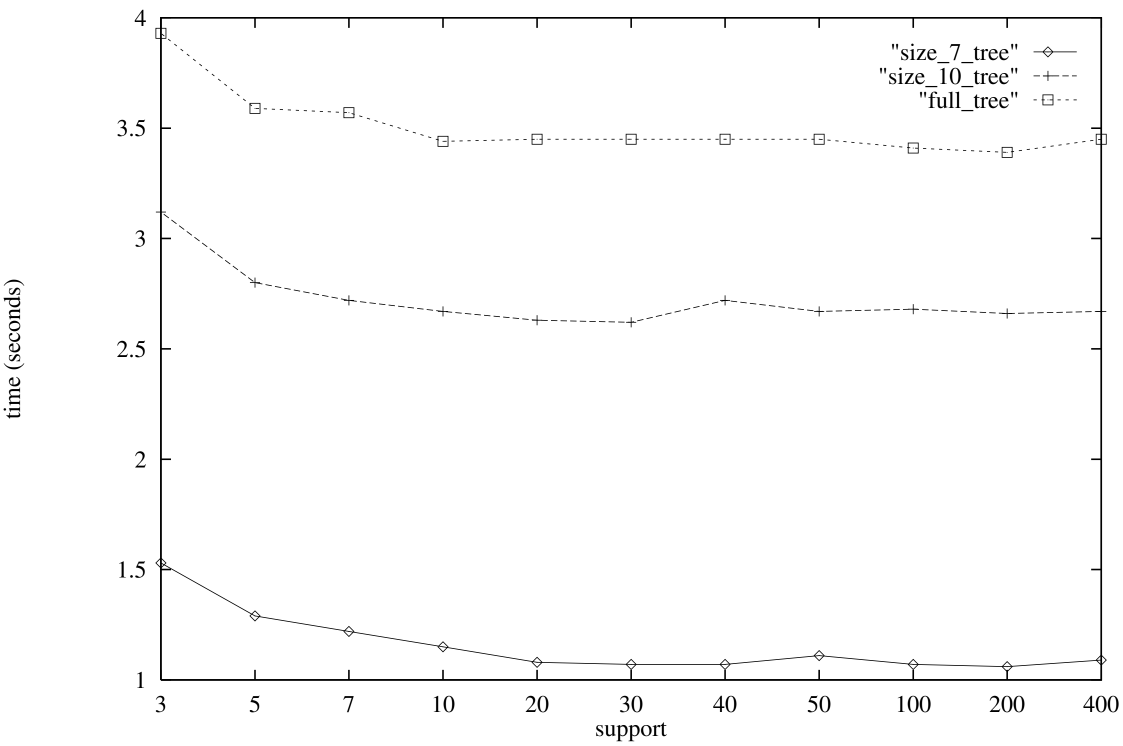
	All Subsets	*Max Cover Size* 10	*Max Cover Size* 7
Space:	4.61 mb	3.57 mb	1.45 mb
Time, construction only:	33.5 sec.	21.8 sec.	10.9 sec.
Time, including save:	35.5 sec.	23.1 sec.	11.6 sec.

In all the following subsections, we start out with an existing trie on disk. The times quoted *include disk access.*

6.2 Generating All Covers

In order to generate all covers, we simply need to do a depth first search (DFS) of the tree, independently of the support. In the graph below one can see the running times for three experiments. In the first, the generated tree was smallest, it was built under the assumption that the number of attributes in a cover is bounded by 7. The second assumed that the number of attributes in a cover is not larger than 10. Finally, we ran the algorithm on the full tree. As can be seen, there is practically no difference in the running time for different supports. The only slight rise is where the support is 3. Note that we are talking about *absolute 3*, every cover that appears even three times was generated! Even the running time for support 3 (0.01%) was very small, 3.93 seconds on the full trie. The reason there is some increment for such a minute support is that now indeed a full DFS took place. Recall that the trie was constructed in a manner where the support decreases on a path from the root to a leaf. Thus, when *minsupp* is large, the tree is only scanned till a node whose support is too small is encountered. Consequently, for larger supports, the DFS does not scan the entire tree, whereas for $minsupp = 3$, the entire trie was scanned by the DFS.

It is important to note that previous algorithms in the literature did not plot such small supports because of their *exponential time growth*. Another important note is that the running time includes disk access to the trie. If the entire trie resides in memory at the start of the algorithm, the running time is several hundred *milliseconds*.

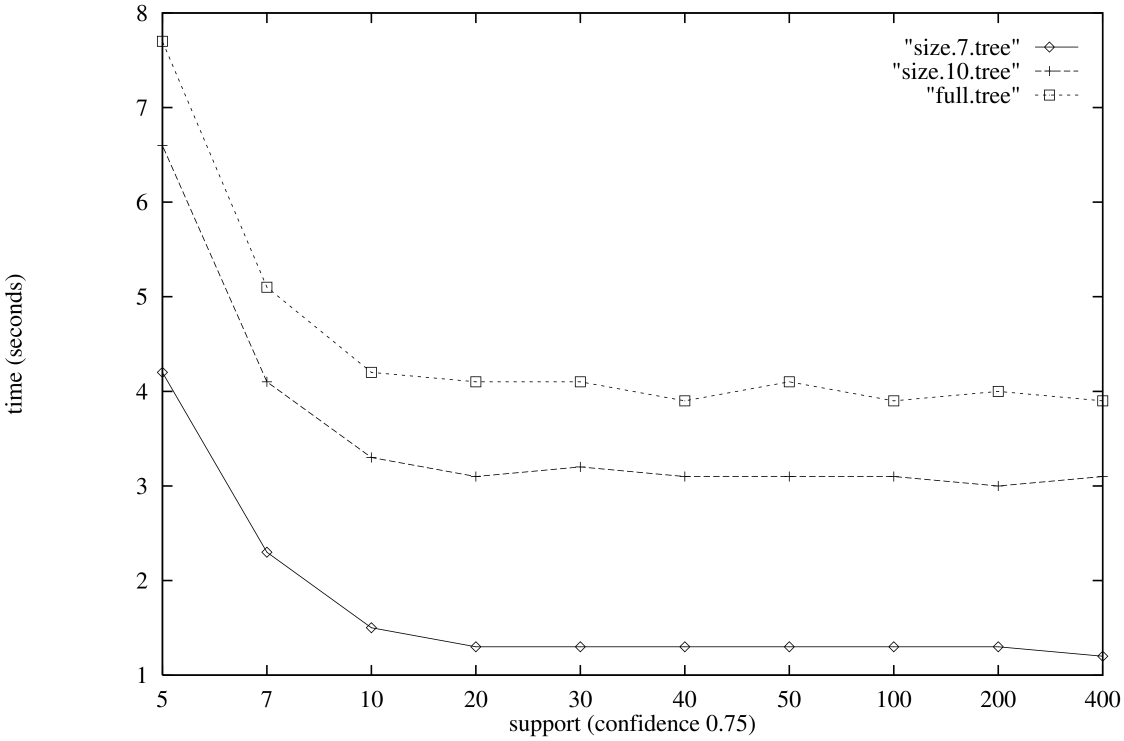


The following table gives the respective running times in seconds.

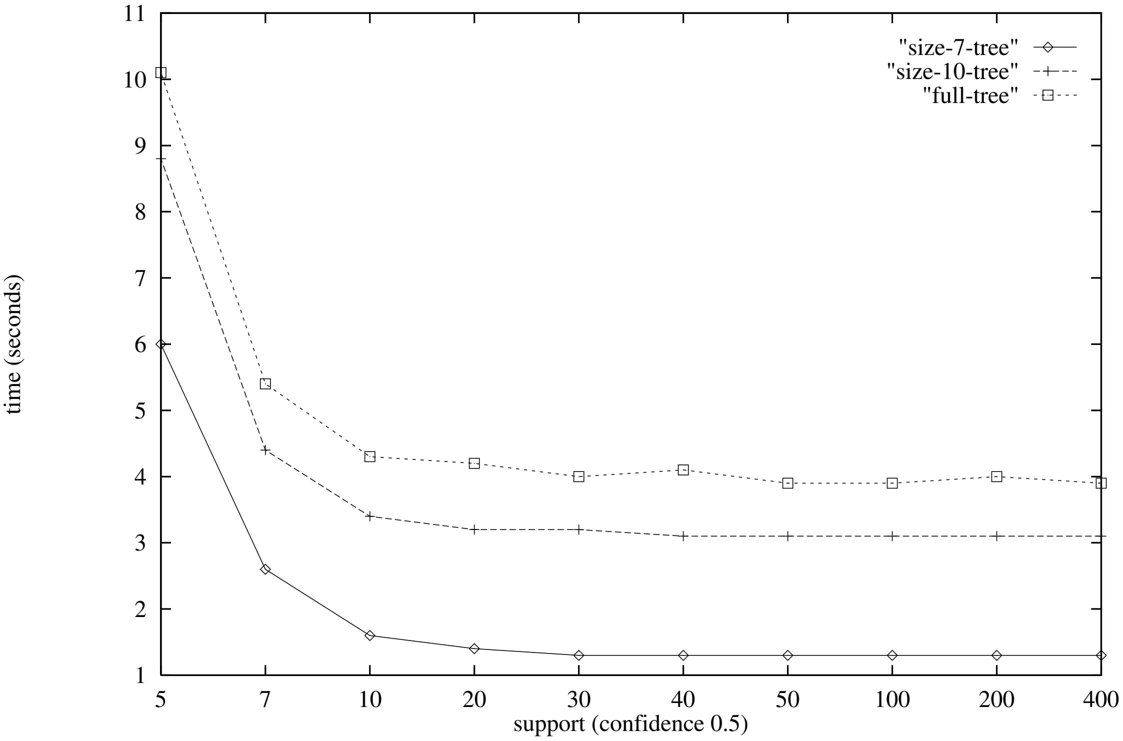
Support	3	5	7	10	20	30	40	50	100	200	400
Full Tree:	3.93	3.59	3.57	3.44	3.45	3.45	3.45	3.45	3.41	3.39	3.45
Size 10 Tree:	3.12	2.80	2.72	2.67	2.63	2.62	2.72	2.67	2.68	2.66	2.67
Size 7 Tree:	1.53	1.29	1.22	1.15	1.08	1.07	1.07	1.11	1.07	1.06	1.09

6.3 Generating all Associations

The following graph represents the time it took to generate all associations with various different supports. The confidence ratio taken in all cases was 75%. As can be seen, the graph shows that the support plays no role in the association generation, except for a slight time increase for ridiculously small supports.



The following graph is similar to the one above but the confidence is taken at 50%. Note that for all reasonable supports (even support 10, which is 0.045%) the results seem to be not only support-independent, but also confidence-independent.



The values in the above two graphs are summarized in the table below. The times given are in seconds. The confidence independence can be seen quite clearly from the data below.

Support:	5	7	10	20	30	40	50	100	200	400
Full Tree (confidence 75%):	7.7	5.1	4.2	4.1	4.1	3.9	4.1	3.9	4.0	3.9
Full Tree (confidence 50%):	10.1	5.4	4.3	4.2	4.0	4.1	3.9	3.9	4.0	3.9
Size 10 Tree (confidence 75%):	6.6	4.1	3.3	3.1	3.2	3.1	3.1	3.1	3.0	3.1
Size 10 Tree (confidence 50%):	8.8	4.4	3.4	3.2	3.2	3.1	3.1	3.1	3.1	3.1
Size 7 Tree (confidence 75%):	4.2	2.3	1.5	1.3	1.3	1.3	1.3	1.3	1.3	1.2
Size 7 Tree (confidence 50%):	6.0	2.6	1.6	1.4	1.3	1.3	1.3	1.3	1.3	1.3

6.4 Generating Excluding Associations

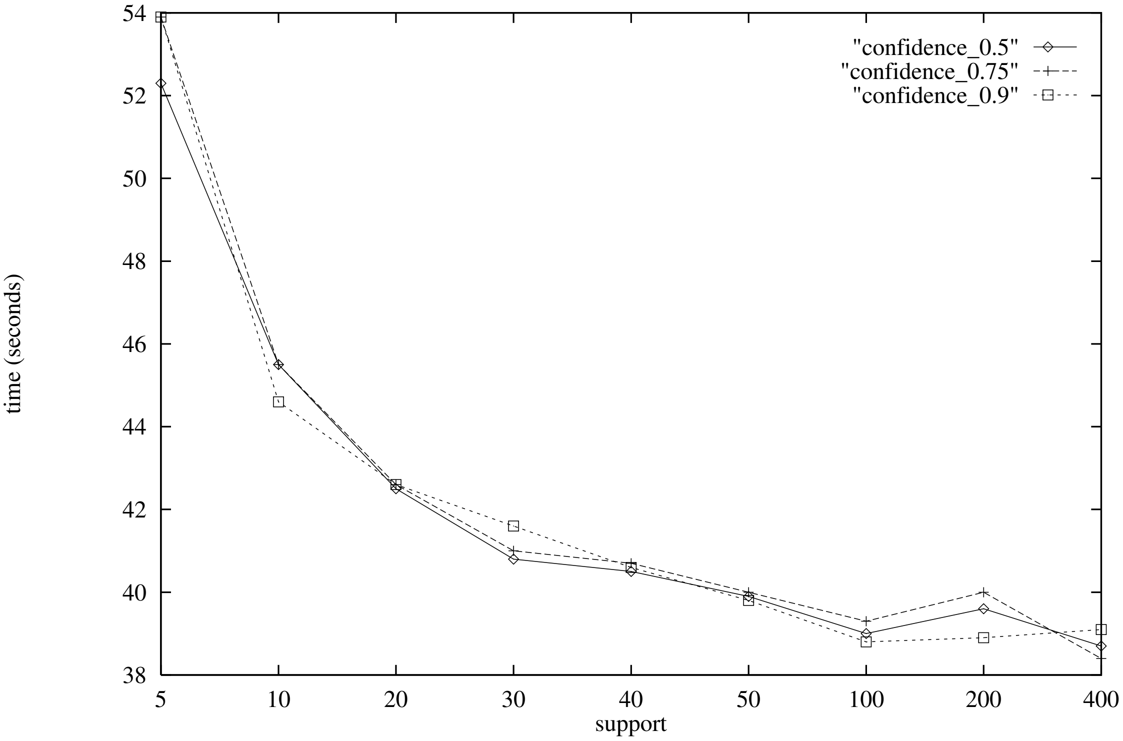
To our knowledge, our algorithm is the only one that can generate excluding associations. The graph below indicates the time it took to generate excluding associations with the different supports. The trie used was the one for maximum cover size 7. The different plots represent the three different confidence ratios used, 90%, 75%, and 50%..

The table below shows the number of associations and the number of excluding associations with a single exclusion and one consequent. The confidence ratio appears as a percentage in parentheses.

It should be noted that all excluding associations we count are new. They are not derived from regular associations. In other words, if $S \Rightarrow \{a\}$ then we do not count $S \cup \{\neg b\} \Rightarrow \{a\}$ for any b not in S. This table shows that computing the excluding associations is quite important since there are a significant number of such associations. In addition, the table also shows the running time of our algorithm to produce all regular associations, and all excluding associations with one exclusion and a single consequent.

Support:	5	10	20	30	40	50	100	200	400
Regular Associations (90%):	7321	733	241	72	34	24	6	3	0
Excluding Associations (90%):	8183	923	215	78	25	11	0	0	0
Time (seconds, conf: 90%):	53.9	44.6	42.6	41.6	40.5	39.8	38.8	38.9	39.1
Regular Associations (75%):	9899	1070	366	144	62	45	13	6	2
Excluding Associations (75%):	7294	921	308	78	26	6	0	0	0
Time (seconds, conf: 75%):	53.9	45.5	42.5	41.0	40.7	40.1	39.3	40.0	38.4
Regular Associations (50%):	13217	1681	617	279	139	91	29	9	3
Excluding Associations (50%):	4329	712	236	121	55	44	5	3	0
Time (seconds, conf: 50%):	52.3	45.5	42.5	40.8	40.5	39.9	39.0	39.6	38.7

The running times are plotted on the following graph.



While the time to generate excluding associations is not nearly as good as the time to generate associations, it is still manageable. To our knowledge, this is the first program that can generate excluding associations.

7 Conclusions and Open Problems

We have shown a new method for generating associations. Our method enables finding associations with extremely small supports, naturally allows finding covers in growing and shrinking databases and is the first known algorithm to generate excluding associations. At the heart of our method was using subword trees for succinct encoding and efficient retrieval of information. Our algorithm performed satisfactorily on the Reuters-22173 database.

From a theoretical point of view, all existing algorithms, including the ones we presented here, have an exponential bottleneck at some stage. It would be an extremely important contribution to develop a method that would only have a polynomial worst-case complexity. We feel that the literature of subword trees may provide answers in this direction. We also feel that further study will allow achieving faster generation of excluding associations.

Acknowledgment: The authors warmly thank an anonymous referee whose insightful comments helped both the presentation and content of the paper.

References

[1] R. Agrawal, T. Imielinski, and A. Swami. Database mining: a performance perspective. *IEEE Trans. Knowledge and Data Engineering*, 5(6):914–925, 1993.

[2] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proc. ACM SIGMOD*, pages 207–216, Washington, DC, May 1993.

[3] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *Proc. 20th Int'l Conf. on VLDB*, Santiago, Chile, Aug 1994.

[4] R. Feldman, A. Amir, Y. Aumann, A. Zilberstein, and H. Hirsh. Incremental algorithms for association generation. to appear, First Pacific Conference on Knowledge Discovery, July 1996.

[5] R. Feldman and I. Dagan. Knowledge discovery in textual databases. *Proc. 1st Intl. Conf. on Knowledge Discovery and Data Mining*, pages 112–117, 1995.

[6] R. Feldman, I. Dagan, and H. Hirsh. Keyword-based browsing and analysis of large document sets. In *Proc. 5th Symp. on Document Analysis and Information Retrieval*, Las Vegas, Nevada, April 1996.

[7] R. Feldman, I. Dagan, and W. Kloesgen. Efficient algorithms for mining and manipulating associations in texts. In *Proc. 13th European Meeting on Cybernetics and Systems Research*, Vienna, Austria, April 1996.

[8] W. Kloesgen. Problems for knowledge discovery in databases and their treatment in the statistical interpreter explora. *Int'l J. for Intelligent Systems*, 7(7):649–673, 1992.

[9] W. Kloesgen. Efficient discovery of interesting statements. *The Journal of Intelligent Information Systems*, 4(1), 1995.

[10] H. Mannila and H. Toivonen. Multiple uses of frequent sets and condensed representations. *Proc. 2nd Int'l Conference on Knowledge Discovery in Databases*, 1996.

[11] H. Mannila, H. Toivonen, and A. I. Verkamo. Efficient algorithms for discovering association rules. *Proc. AAAI Workshop on Knowledge Discovery in Databases*, pages 181–192, 1994.

[12] G. Piatetsky-Shapiro and W. J. Frawley, editors. *Knowledge Discovery in Databases*. AAAI Press/MIT Press, 1994.

[13] A. Savasere, E. Omiecinski, and S. Navathe. An efficient algorithm for mining association rules in large databases. *Proc. 21st Int'l Conf. on VLDB*, 1995.

[14] R. Sedgewick. *Algorithms*. Addison-Wesley, second edition, 1988.