Mining Generalized Association Rules

Ramakrishnan Srikant* Rakesh Agrawal

IBM Almaden Research Center
San Jose, CA 95120
{srikant,ragrawal}@almaden.ibm.com

Abstract

We introduce the problem of mining generalized association rules. Given a large database of transactions, where each transaction consists of a set of items, and a taxonomy (*is-a* hierarchy) on the items, we find associations between items at any level of the taxonomy. For example, given a taxonomy that says that jackets *is-a* outerwear *is-a* clothes, we may infer a rule that "people who buy outerwear tend to buy shoes". This rule may hold even if rules that "people who buy jackets tend to buy shoes", and "people who buy clothes tend to buy shoes" do not hold. An obvious solution to the problem is to add all ancestors of each item in a transaction to the transaction, and then run any of the algorithms for mining association rules on these "extended transactions". However, this "Basic" algorithm is not very fast; we present two algorithms, Cumulate and EstMerge, which run 2 to 5 times faster than Basic (and more than 100 times faster on one real-life dataset). We also present a new interest-measure for rules which uses the information in the taxonomy. Given a user-specified "minimum-interest-level", this measure prunes a large number of redundant rules; 40% to 60% of all the rules were pruned on two real-life datasets.

*Also, Department of Computer Science, University of Wisconsin, Madison.

Proceedings of the 21st VLDB Conference
Zurich, Swizerland, 1995

1 Introduction

Data mining, also known as knowledge discovery in databases, has been recognized as a new area for database research. The area can be defined as efficiently discovering interesting rules from large collections of data.

The problem of mining association rules was introduced in [1]. Given a set of transactions, where each transaction is a set of items, an association rule is an expression $X \Rightarrow Y$, where X and Y are sets of items. The intuitive meaning of such a rule is that transactions in the database which contain the items in X tend to also contain the items in Y. An example of such a rule might be that 98% of customers who purchase tires and auto accessories also buy some automotive services; here 98% is called the *confidence* of the rule. The *support* of the rule $X \Rightarrow Y$ is the percentage of transactions that contain both X and Y. The problem of mining association rules is to find all rules that satisfy a user-specified minimum support and minimum confidence. Applications include cross-marketing, attached mailing, catalog design, loss-leader analysis, store layout, and customer segmentation based on buying patterns.

In most cases, taxonomies (*is-a* hierarchies) over the items are available. An example of a taxonomy is shown in Figure 1: this taxonomy says that Jacket *is-a* Outerwear, Ski Pants *is-a* Outerwear, Outerwear *is-a* Clothes, etc. Users are interested in generating rules that span different levels of the taxonomy. For example, we may infer a rule that people who buy Outerwear tend to buy Hiking Boots from the fact that people bought Jackets with Hiking Boots and and Ski Pants with Hiking Boots. However, the support for the rule "Outerwear $\Rightarrow$ Hiking Boots" may not be the sum of the supports for the rules "Jackets $\Rightarrow$ Hiking Boots" and "Ski Pants $\Rightarrow$ Hiking Boots" since some people may have bought Jackets, Ski Pants and Hiking Boots in the same transaction. Also, "Outerwear $\Rightarrow$ Hiking Boots" may be a valid rule, while "Jackets $\Rightarrow$ Hiking Boots" and "Clothes $\Rightarrow$ Hiking Boots" may

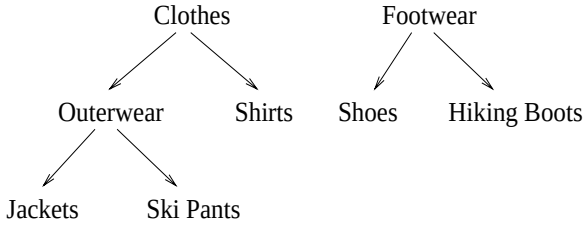

Figure 1: Example of a Taxonomy

not. The former may not have minimum support, and the latter may not have minimum confidence.

Earlier work on association rules [1] [2] [5] [6] [7] did not consider the presence of taxonomies and restricted the items in association rules to the leaf-level items in the taxonomy. However, finding rules across different levels of the taxonomy is valuable since:

- Rules at lower levels may not have minimum support. Few people may buy Jackets with Hiking Boots, but many people may buy Outerwear with Hiking Boots. Thus many significant associations may not be discovered if we restrict rules to items at the leaves of the taxonomy. Since department stores or supermarkets typically have hundreds of thousands of items, the support for rules involving only leaf items (typically UPC or SKU codes) tends to be extremely small.

- Taxonomies can be used to prune uninteresting or redundant rules. We will discuss this further in Section 2.1.

Multiple taxonomies may be present. For example, there could be a taxonomy for the price of items (cheap, expensive, etc.), and another for the category. Multiple taxonomies may be modeled as a single taxonomy which is a DAG (directed acyclic graph). A common application that uses multiple taxonomies is loss-leader analysis. In addition to the usual taxonomy which classifies items into brands, categories, product groups, etc., there is a second taxonomy where items which are on sale are considered to be children of a "items-on-sale" category, and users look for rules containing the "items-on-sale" item.

In this paper, we introduce the problem of mining *generalized* association rules. Informally, given a set of transactions and a taxonomy, we want to find association rules where the items may be from any level of the taxonomy. We give a formal problem description in Section 2. One drawback users experience in applying association rules to real problems is that they tend to get a lot of uninteresting or redundant rules along with the interesting rules. We introduce an interest-measure that uses the taxonomy to prune redundant rules.

An obvious solution to the problem is to replace each transaction T with an "extended transaction" T', where T' contains all the items in T as well as all the ancestors of each items in T. For example, if the transaction contained Jackets, we would add Outerwear and Clothes to get the extended-transaction. We can then run any of the algorithms for mining association rules [1] [2] [5] [6] [7] on the extended transactions to get generalized association rules. However, this "Basic" algorithm is not very fast; two more sophisticated algorithms that we propose run 2 to 5 times faster than Basic (and more than 100 times faster on one real-life dataset).

We describe the Basic algorithm and our two algorithms in Section 3, and evaluate their performance on both synthetic and real-life data in Section 4. Finally, we summarize our work and conclude in Section 5. For an expanded version of this paper, see [9].

2 Problem Statement

Let $\mathcal{I} = \{i_1, i_2, \ldots, i_m\}$ be a set of literals, called items. Let $\mathcal{T}$ be a directed acyclic graph on the literals. An edge in $\mathcal{T}$ represents an *is-a* relationship, and $\mathcal{T}$ represents a set of taxonomies. If there is an edge in $\mathcal{T}$ from p to c, we call p a *parent* of c and c a *child* of p. (p represents a generalization of c.) We model the taxonomy as a DAG rather than a forest to allow for multiple taxonomies.

We use lower case letters to denote items and upper case letters for sets of items (itemsets). We call $\widehat{x}$ an *ancestor* of x (and x a *descendant* of $\widehat{x}$) if there is an edge from $\widehat{x}$ to x in the transitive-closure of $\mathcal{T}$. Note that a node is not an ancestor of itself, since the graph is acyclic.

Let $\mathcal{D}$ be a set of transactions, where each transaction T is a set of items such that $T \subseteq \mathcal{I}$. (While we expect the items in T to be leaves in $\mathcal{T}$, we do not require this.) We say that a transaction T *supports* an item $x \in \mathcal{I}$ if x is in T or x is an ancestor of some item in T. We say that a transaction T *supports* $X \subseteq \mathcal{I}$ if T supports every item in X.

A *generalized association rule* is an implication of the form $X \Rightarrow Y$, where $X \subset \mathcal{I}$, $Y \subset \mathcal{I}$, $X \cap Y = \emptyset$, and no item in Y is an ancestor of any item in X. The rule $X \Rightarrow Y$ holds in the transaction set $\mathcal{D}$ with *confidence* c if c% of transactions in $\mathcal{D}$ that support X also support Y. The rule $X \Rightarrow Y$ has *support* s in the transaction set $\mathcal{D}$ if s% of transactions in $\mathcal{D}$ support $X \cup Y$. The reason for the condition that no item in Y should be an ancestor of any item in X is that a rule of the form "x $\Rightarrow$ ancestor(x)" is trivially true with 100% confidence, and hence redundant. We call these rules generalized association rules because both X and Y can contain items from any level of

the taxonomy $\mathcal{T}$, a possibility not entertained by the formalism introduced in [1].

Problem Statement (Tentative) Given a set of transactions $\mathcal{D}$ and a set of taxonomies $\mathcal{T}$, the problem of mining generalized association rules is to discover all rules that have support and confidence greater than the user-specified minimum support (called *minsup*) and minimum confidence (called *minconf*) respectively.

This definition has the problem that many "redundant" rules may be found. We will formalize the notion of redundancy and modify the problem statement accordingly in Section 2.1. (We introduce the tentative problem statement here in order to explain redundancy.)

Example Let $\mathcal{I}$ = {Footwear, Shoes, Hiking Boots, Clothes, Outerwear, Jackets, Ski Pants, Shirts} and $\mathcal{T}$ the taxonomy shown in Figure 1. Consider the database shown in Figure 2. Let minsup be 30% (that is, 2 transactions) and minconf 60%. Then the sets of items with minimum support (*frequent* itemsets), and the rules corresponding to the these itemsets are shown in Figure 2. Note that the rules "Ski Pants ⇒ Hiking Boots" and "Jackets ⇒ Hiking Boots" do not have minimum support, but the rule "Outerwear ⇒ Hiking Boots" does.

Observation Let $\Pr(X)$ denote the probability that *all* the items in X are contained in a transaction. Then support($X \Rightarrow Y$) = $\Pr(X \cup Y)$ and confidence($X \Rightarrow Y$) = $\Pr(Y \mid X)$. (Note that $\Pr(X \cup Y)$ is the probability that all the items in $X \cup Y$ are present in the transaction.)

If a set $\{x,y\}$ has minimum support, so do $\{x,\widehat{y}\}$, $\{\widehat{x},y\}$ and $\{\widehat{x},\widehat{y}\}$. ($\widehat{x}$ denote an ancestor of x). However if the rule $x \Rightarrow y$ has minimum support and confidence, only the rule $x \Rightarrow \widehat{y}$ is guaranteed to have both minimum support and confidence. While the rules $\widehat{x} \Rightarrow y$ and $\widehat{x} \Rightarrow \widehat{y}$ will have minimum support, they may not have minimum confidence.

The support for an item in the taxonomy is *not* equal to the sum of the supports of its children, since several of the children could be present in a single transaction. Hence we cannot directly infer rules about items at higher levels of the taxonomy from rules about the leaves.

2.1 Interesting Rules

Previous work on quantifying the "usefulness" or "interest" of a rule focussed on how much the support of a rule was more than the expected support based on the support of the antecedent and consequent. In [8], Piatetsky-Shapiro argues that a rule $X \Rightarrow Y$ is not interesting if support($X \Rightarrow Y$) $\approx$ support(X) $\times$ support(Y). We implemented this idea, and used the chi-square value to check if the rule was statistically significant. However, this measure did not prune many rules; on two real-life datasets (described in Section 4.5), less than 1% of the rules were found to be redundant (not statistically significant). In this section, we use the information in taxonomies to derive a new interest measure that prunes out 40% to 60% of the rules as "redundant" rules.

Database $\mathcal{D}$

Transaction	Items Bought
100	Shirt
200	Jacket, Hiking Boots
300	Ski Pants, Hiking Boots
400	Shoes
500	Shoes
600	Jacket

Frequent Itemsets

Itemset	Support
{ Jacket }	2
{ Outerwear }	3
{ Clothes }	4
{ Shoes }	2
{ Hiking Boots }	2
{ Footwear }	4
{ Outerwear, Hiking Boots }	2
{ Clothes, Hiking Boots }	2
{ Outerwear, Footwear }	2
{ Clothes, Footwear }	2

Rules

Rule	Support	Conf.
Outerwear ⇒ Hiking Boots	33%	66.6%
Outerwear ⇒ Footwear	33%	66.6%
Hiking Boots ⇒ Outerwear	33%	100%
Hiking Boots ⇒ Clothes	33%	100%

Figure 2: Example

To motivate our approach, consider the rule

Milk ⇒ Cereal (8% support, 70% confidence)

If "Milk" is a parent of "Skim Milk", and about a quarter of sales of "Milk" are "Skim Milk", we would expect the rule

Skim Milk ⇒ Cereal

to have 2% support and 70% confidence. If the actual support and confidence for "Skim Milk ⇒ Cereal" are around 2% and 70% respectively, the rule can be

considered redundant since it does not convey any additional information and is less general than the first rule. We capture this notion of "interest" by saying that we only want to find rules whose support is more than R times the expected value or whose confidence is more than R times the expected value, for some user-specified constant R.[1] We formalize the above intuition below.

We call $\widehat{Z}$ an *ancestor* of Z (where $Z, \widehat{Z}$ are sets of items such that $Z, \widehat{Z} \subseteq \mathcal{I}$) if we can get $\widehat{Z}$ from Z by replacing one or more items in Z with their ancestors and Z and $\widehat{Z}$ have the same number of items. (The reason for the latter condition is that it is not meaningful to compute the expected support of Z from $\widehat{Z}$ unless they have the same number of items. For instance, the support for {Clothes} does give any clue about the expected support for {Outerwear, Shirts}.) We call the rules $\widehat{X} \Rightarrow Y$, $\widehat{X} \Rightarrow \widehat{Y}$ or $X \Rightarrow \widehat{Y}$ ancestors of the rule $X \Rightarrow Y$. Given a set of rules, we call $\widehat{X} \Rightarrow \widehat{Y}$ a *close ancestor* of $X \Rightarrow Y$ if there is no rule $X' \Rightarrow Y'$ such that $X' \Rightarrow Y'$ is an ancestor of $X \Rightarrow Y$ and $\widehat{X} \Rightarrow \widehat{Y}$ is an ancestor of $X' \Rightarrow Y'$. (Similar definitions apply for $X \Rightarrow \widehat{Y}$ and $\widehat{X} \Rightarrow Y$.)

Consider a rule $X \Rightarrow Y$, and let $Z = X \cup Y$. The support of Z will be the same as the support of the rule $X \Rightarrow Y$. Let $E_{\widehat{Z}}[\Pr(Z)]$ denote the "expected" value of $\Pr(Z)$ given $\Pr(\widehat{Z})$, where $\widehat{Z}$ is an ancestor of Z. Let $Z = \{z_1, \ldots, z_n\}$ and $\widehat{Z} = \{\widehat{z_1}, \ldots, \widehat{z_j}, z_{j+1}, \ldots, z_n\}$, $1 \leq j \leq n$, where $\widehat{z_i}$ is an ancestor of z_i. Then we define

$$E_{\widehat{Z}}[\Pr(Z)] = \frac{\Pr(z_1)}{\Pr(\widehat{z_1})} \times \cdots \times \frac{\Pr(z_j)}{\Pr(\widehat{z_j})} \times \Pr(\widehat{Z}).$$

to be the expected value of $\Pr(Z)$ given the itemset $\widehat{Z}$.[2]

[1] We can easily enhance this definition to say that we want to find rules with minimum support whose support (or confidence) is either more or less than the expected value. However, many rules whose support is less than expected will not have minimum support. In fact, the more the deviation from the expected value, the less the support for the rule. So the most interesting rules may not have minimum support. (The same applies for confidence.) Suppose we wanted to find all rules whose support is less than expected, irrespective of minimum support. Consider a "typical" database with 50,000 items, an average of 5 items per transaction and ten million transactions. The average probability that an item is present in a transaction is 1/10,000; that any two items are present in the same transaction 1/100,000,000. Hence, on average, the expected number of transactions where two specific items are bought together is just 0.1. There may be millions of rules which say that two items are never bought together, and these rules would not even be significant.

[2] Alternate definitions are possible. For example, we could define:

$$E_{\widehat{Z}}[\Pr(Z)] = \frac{\Pr(\{z_1, \ldots, z_j\})}{\Pr(\{\widehat{z_1}, \ldots, \widehat{z_j}\})} \times \Pr(\widehat{Z}).$$

Similarly, let $E_{\widehat{X} \Rightarrow \widehat{Y}}[\Pr(Y \mid X)]$ denote the "expected" confidence of the rule $X \Rightarrow Y$ given the rule $\widehat{X} \Rightarrow \widehat{Y}$. Let $Y = \{y_1, \ldots, y_n\}$ and $\widehat{Y} = \{\widehat{y_1}, \ldots, \widehat{y_j}, y_{j+1}, \ldots, y_n\}$, $1 \leq j \leq n$, where $\widehat{y_i}$ is an ancestor of y_i. Then we define

$$E_{\widehat{X} \Rightarrow \widehat{Y}}[\Pr(Y \mid X)] = \frac{\Pr(y_1)}{\Pr(\widehat{y_1})} \times \cdots \times \frac{\Pr(y_j)}{\Pr(\widehat{y_j})} \times \Pr(\widehat{Y} \mid \widehat{X})$$

Note that $E_{\widehat{X} \Rightarrow Y}[\Pr(Y \mid X)] = \Pr(Y \mid \widehat{X})]$.

We call a rule $X \Rightarrow Y$ *R-interesting* w.r.t an ancestor $\widehat{X} \Rightarrow \widehat{Y}$ if the support of the rule $X \Rightarrow Y$ is R times the expected support based on $\widehat{X} \Rightarrow \widehat{Y}$, or the confidence is R times the expected confidence based on $\widehat{X} \Rightarrow \widehat{Y}$.

Definition of Interesting Rules Given a set of rules S and a minimum interest R, a rule $X \Rightarrow Y$ is *interesting* (in S) if it has no ancestors or it is R-interesting with respect to its close ancestors among its interesting ancestors. We say that an rule $X \Rightarrow Y$ is *partially interesting* (in S) if it has no ancestors or is R-interesting with respect to at least one close ancestor among its interesting ancestors.

We motivate the reason for only considering close ancestors among all interesting ancestors with an example. Consider the rules shown in Figure 3. The support for the items in the antecedent are shown alongside. Assume we have the same taxonomy as in the previous example. Rule 1 has no ancestors and is hence interesting. The support for rule 2 is twice the expected support based on rule 1, and is thus interesting. The support for rule 3 is exactly the expected support based on rule 2, but twice the support based on rule 1. We do not want consider rule 3 to be interesting since its support can be predicted based on rule 2, even though its support is more than expected if we ignore rule 2 and look at rule 1.

2.2 Problem Statement

Given a set of transactions $\mathcal{D}$ and a user-specified minimum interest (called *min-interest*), the problem of mining association rules with taxonomies is to find all interesting association rules that have support and confidence greater than the user-specified minimum support (called *minsup*) and minimum confidence (called *minconf*) respectively.

For some applications, we may want to find partially interesting rules rather than just interesting rules. Note that if min-interest = 0, all rules are found, regardless of interest.

3 Algorithms

The problem of discovering generalized association rules can be decomposed into three parts:

Rule #	Rule	Support
1	"Clothes ⇒ Footwear"	10
2	"Outerwear ⇒ Footwear"	8
3	"Jackets ⇒ Footwear"	4

Item	Support
Clothes	5
Outerwear	2
Jackets	1

Figure 3: Example - Interest

1. Find all sets of items (*itemsets*) whose support is greater than the user-specified minimum support. Itemsets with minimum support are called *frequent* itemsets.[3]

2. Use the frequent itemsets to generate the desired rules. The general idea is that if, say, $ABCD$ and AB are frequent itemsets, then we can determine if the rule $AB \Rightarrow CD$ holds by computing the ratio $conf = \text{support}(ABCD)/\text{support}(AB)$. If $conf \geq minconf$, then the rule holds. (The rule will have minimum support because $ABCD$ is frequent.)

3. Prune all uninteresting rules from this set.

In the rest of this section, we look at algorithms for finding all frequent itemsets where the items can be from any level of the taxonomy. Given the frequent itemsets, the algorithm in [1] [2] can be used to generate rules. We first describe the obvious approach for finding frequent itemsets, and then present our two algorithms.

3.1 Algorithm Basic

Consider the problem of deciding whether a transaction T supports an itemset X. If we take the raw transaction, this involves checking for each item $x \in X$ whether x or some descendant of x is present in the transaction. The task become much simpler if we first add all the ancestors of each item in T to T; let us call this extended transaction T'. Now T supports X if and only if T' is a superset of X. Hence a straight-forward way to find generalized association rules would be to run any of the algorithms for finding association rules from [1] [2] [5] [6] [7] on the extended transactions. We discuss below the generalization of the Apriori algorithm given in [2]. Figure 5 gives an overview of the algorithm, using the notation in Figure 4.

The first pass of the algorithm simply counts item occurrences to determine the frequent 1-itemsets. Note that items in the itemsets can come from the leaves of the taxonomy or from interior nodes. A subsequent pass, say pass k, consists of two phases. First,

[3] In earlier papers [1] [2], itemsets with minimum support were called *large* itemsets. However, some readers associated "large" with the number of items in the itemset, rather than its support. So we are switching the terminology to *frequent* itemsets.

k-itemset	An itemset having k items.
L_k	Set of frequent k-itemsets (those with minimum support).
C_k	Set of candidate k-itemsets (potentially frequent itemsets).
$\mathcal{T}$	Taxonomy

Figure 4: Notation

L_1 := {frequent 1-itemsets};
$k := 2$; // k represents the pass number
while ($L_{k-1} \neq \emptyset$) **do**
begin
  C_k := New candidates of size k generated from L_{k-1}.
  forall transactions $t \in \mathcal{D}$ **do**
  begin
    Add all ancestors of each item in t to t, removing any duplicates.
    Increment the count of all candidates in C_k that are contained in t.
  end
  L_k := All candidates in C_k with minimum support.
  $k := k + 1$;
end
Answer := $\bigcup_k L_k$;

Figure 5: Algorithm Basic

the frequent itemsets L_{k-1} found in the $(k-1)$th pass are used to generate the candidate itemsets C_k, using the apriori candidate generation function described in the next paragraph. Next, the database is scanned and the support of candidates in C_k is counted. For fast counting, we need to efficiently determine the candidates in C_k that are contained in a given transaction t. We reuse the hash-tree data structure described in [2] for this purpose.

Candidate Generation Given L_{k-1}, the set of all frequent $(k-1)$-itemsets, we want to generate a superset of the set of all frequent k-itemsets. Candidates may include leaf-level items as well as interior nodes in the taxonomy. The intuition behind this procedure is that if an itemset X has minimum support, so do all subsets of X. For simplicity, assume the items in each itemset are kept sorted in lexicographic order. First, in the *join* step, we join L_{k-1} with L_{k-1}:

insert into C_k
select $p.\text{item}_1$, $p.\text{item}_2$, ..., $p.\text{item}_{k-1}$, $q.\text{item}_{k-1}$

from L_{k-1} p, L_{k-1} q
where $p.\text{item}_1 = q.\text{item}_1$, ..., $p.\text{item}_{k-2} = q.\text{item}_{k-2}$, $p.\text{item}_{k-1} < q.\text{item}_{k-1}$;

Next, in the *prune* step, we delete all itemsets $c \in C_k$ such that some $(k-1)$-subset of c is not in L_{k-1}.

3.2 Algorithm Cumulate

We add several optimizations to the Basic algorithm to develop the algorithm "Cumulate". The name indicates that all itemsets of a certain size are counted in one pass, unlike the "Stratify" algorithm in Section 3.3.

1. **Filtering the ancestors added to transactions.** We do not have to add all ancestors of the items in a transaction t to t. Instead, we just need to add ancestors that are in one (or more) of the candidate itemsets being counted in the current pass. In fact, if the original item is not in any of the itemsets, it can be dropped from the transaction.

   For example, assume the parent of "Jacket" is "Outerwear", and the parent of "Outerwear" is "Clothes". Let {Clothes, Shoes} be the only itemset being counted. Then, in any transaction containing Jacket, we simply replace Jacket by Clothes. We do not need to keep Jacket in the transaction, nor do we need to add Outerwear to the transaction.

2. **Pre-computing ancestors.** Rather than finding ancestors for each item by traversing the taxonomy graph, we can pre-compute the ancestors for each item. We can drop ancestors that are not present in any of the candidates at the same time.

3. **Pruning itemsets containing an item and its ancestor.** We first present two lemmas to justify this optimization.

   Lemma 1 *The support for an itemset X that contains both an item x and its ancestor $\hat{x}$ will be the same as the support for the itemset $X - \hat{x}$.*

   Lemma 2 *If L_k, the set of frequent k-itemsets, does not include any itemset that contains both an item and its ancestor, the set of candidates C_{k+1} generated by the candidate generation procedure in Section 3.1 will not include any itemset that contains both an item and its ancestor.*

   Proofs of these lemmas are given in [9]. Lemma 1 shows that we need not count any itemset which contains both an item and its ancestor. We add this optimization by pruning the candidate itemsets of size two which consist of an item and its ancestor. Lemma 2 shows that pruning these candidates is sufficient to ensure that we never generate candidates in subsequent passes which contain both an item and its ancestor.

```
Compute T*, the set of ancestors of each item,
    from T.  // Optimization 2
L_1 := {frequent 1-itemsets};
k := 2;  // k represents the pass number
while ( L_{k-1} ≠ ∅ ) do
begin
    C_k := New candidates of size k generated from L_{k-1}.
    if (k = 2) then
        Delete any candidate in C_2 that consists of an
            item and its ancestor.  // Optimization 3
    Delete any ancestors in T* that are not present in
        any of the candidates in C_k.  // Optimization 1
    forall transactions t ∈ D do
    begin
        foreach item x ∈ t do
            Add all ancestors of x in T* to t.
        Remove any duplicates from t.
        Increment the count of all candidates in C_k
            that are contained in t.
    end
    L_k := All candidates in C_k with minimum support.
    k := k + 1;
end
Answer := ⋃_k L_k;
```

Figure 6: Algorithm Cumulate

Figure 6 gives an overview of the Cumulate algorithm.

3.3 Stratification

We motivate this algorithm with an example. Let {Clothes, Shoes}, {Outerwear, Shoes} and {Jacket, Shoes} be candidate itemsets to be counted, with "Jacket" being the child of "Outerwear", and "Outerwear" the child of "Clothes". If {Clothes, Shoes} does not have minimum support, we do not have to count either {Outerwear, Shoes} or {Jacket, Shoes}. Thus, rather than counting all candidates of a given size in the same pass as in Cumulate, it may be faster to first count the support of {Clothes, Shoes}, then count {Outerwear, Shoes} if {Clothes, Shoes} turns out to have minimum support, and finally count {Jacket, Shoes} if {Outerwear, Shoes} also has minimum support. Of course, the extra cost in making multiple passes over the database may be more than the benefit of counting fewer itemsets. We will discuss this tradeoff in more detail shortly.

We develop this algorithm by first presenting the straight-forward version, "Stratify", and then describ-

ing the use of sampling to increase its effectiveness (the "Estimate" and "EstMerge" versions). The optimizations we introduced for the Cumulate algorithm apply to this algorithm as well.

3.3.1 Stratify

Consider the partial ordering induced by the taxonomy DAG on a set of itemsets. Itemsets with no parents are considered to be at depth 0. For other itemsets, the depth of an itemset X is defined to be $(\max(\{\text{depth}(\widehat{X}) \mid \widehat{X} \text{ is a parent of } X\}) + 1)$.

We first count all itemsets C_0 at depth 0. After deleting candidates that are descendants of those itemsets in C_0 that did not have minimum support, we count the remaining itemsets at depth 1 (C_1). After deleting candidates that are descendants of the itemsets in C_1 without minimum support, we count the itemsets at depth 2, etc. If there are only a few candidates at depth n, we can count candidates at different depths (n, $n+1$, ...) together to reduce the overhead of making multiple passes.

There is a tradeoff between the number of itemsets counted (CPU time) and the number of passes over the database (IO+CPU time). One extreme would be to make a pass over the database for the candidates at each depth. This would result in a minimal number of itemsets being counted, but we may waste a lot of time in scanning the database multiple times. The other extreme would be to make just one pass for all the candidates, which is what Cumulate does. This would result in counting many itemsets that do not have minimum support and whose parents do not have minimum support. In our implementation, we used the heuristic (empirically determined) that we should count at least 20% of the candidates in each pass.

3.3.2 Estimate

Rather than hoping that candidates which include items at higher levels of the taxonomy will not have minimum support, resulting in our not having to count candidates which include items at lower levels, we can use sampling to estimate the support of candidates. We then count candidates that are expected to have minimum support as well as candidates that are not expected to have minimum support but all of whose parents have minimum support. (We call this set C'_k, for candidates of size k.) We expect that the latter candidates will not have minimum support, and hence we will not have to count any of the descendants of those candidates. If some of those candidates turn out to have minimum support support, we make an extra pass to count their descendants. (We call this set of candidates C''_k.) If we only count candidates that are expected to have minimum support, we will have to make another pass to count their children, since we can only be sure that their children do not have minimum support if we actually count them.

In our implementation, we included candidates whose support in the sample was 0.9 times the minimum support, and candidates all of whose parents had 0.9 times the minimum support, in C'_k in order to reduce the effect of sampling error. We will discuss the effect of changing this sampling error margin shortly, when we also discuss how the sample size can the chosen.

Example For example, consider the three candidates shown in Figure 7. Let "Jacket" be a child of "Outerwear" and "Outerwear" a child of "Clothes". Let minimum support be 5%, and let the support for the candidates in a sample of the database be as shown in Figure 7. Hence, based on the sample, we expect only {Clothes, Shoes} to have minimum support over the database. We now find the support of *both* {Clothes, Shoes} and {Outerwear, Shoes} over the entire database. We count {Outerwear, Shoes} even though we do not expect it to have minimum support since we will not know for sure whether it has minimum support unless {Clothes, Shoes} does not have minimum support, and we expect {Clothes, Shoes} to have minimum support. Now, in scenario A, we do not have to find the support for {Jacket, Shoes} since {Outerwear, Shoes} does not have minimum support (over the entire database). However, in scenario B, we have to make an extra pass to count {Jacket, Shoes}.

3.3.3 EstMerge

Since the estimate (based on the sample) of which candidates have minimum support has some error, Estimate usually makes a second pass where it counts the support for the candidates in C''_k (the descendants of candidates in C_k that were wrongly expected to not have minimum support.) The number of candidates counted in this pass is usually small. Rather than making a separate pass to count these candidates, we can count them when we count candidates in C_{k+1}. However, since we do not know if the candidates in C''_k will have minimum support or not, we assume all these candidates to be frequent when generating C_{k+1}. That is, we will consider L_k to be those candidates in C'_k with minimum support, as well as all candidates in C''_k, when generating C_{k+1}. This can generate more candidates in C_{k+1} than would be generated by Estimate, but does not affect correctness. The tradeoff is between the extra candidates counted by EstMerge against the extra pass made by Estimate. An overview of the algorithm is given in Figure 8. (All the optimizations introduced for the Cumulate algorithm ap-

Candidate Itemsets	Support in Sample	Support in Database	
		Scenario A	Scenario B
{Clothes, Shoes}	8%	7%	9%
{Outerwear, Shoes}	4%	4%	6%
{Jacket, Shoes}	2%		

Figure 7: Example for Estimate

L_1 := {frequent 1-itemsets};
Generate $\mathcal{D}_S$, a sample of the database, in the first pass;
$k := 2$; // k represents the pass number
$C''_1 := \emptyset$; // C''_k represents candidates of size k to
// be counted with candidates of size $k+1$
while ($L_{k-1} \neq \emptyset$ **or** $C''_{k-1} \neq \emptyset$) **do**
begin
  C_k := New candidates of size k generated from $L_{k-1} \cup C''_{k-1}$.
  Estimate the support of the candidates in C_k by making a pass over $\mathcal{D}_S$.
  C'_k := Candidates in C_k that are expected to have minimum support and candidates all of whose parents are expected to have minimum support.
  Find the support of the candidates in $C'_k \cup C''_{k-1}$ by making a pass over $\mathcal{D}$.
  Delete all candidates in C_k whose ancestors (in C'_k) do not have minimum support.
  C''_k := Remaining candidates in C_k that are not in C'_k.
  L_k := All candidates in C'_k with minimum support.
  Add all candidates in C''_{k-1} with minimum support to L_{k-1}.
  $k := k + 1$;
end
Answer := $\bigcup_k L_k$;

Figure 8: Algorithm EstMerge

ply here, though we have omitted them in the figure.)

3.3.4 Size of Sample

We now discuss how to select the sample size for estimating the support of candidates. Let p be the support (as a fraction) of a given itemset X. Consider a random sample with replacement of size n from the database. Then the number of transactions in the sample that contain X is a random variable s with binomial distribution of n trials, each having success probability p. We use the abbreviation $s \succeq k$ ("s is at least as extreme as k") defined by

$$s \succeq k \Longleftrightarrow \begin{cases} x \geq k & \text{if } k \geq pn \\ x \leq k & \text{if } k < pn \end{cases}$$

Using Chernoff bounds [4] [3], the probability that the fractional support in the sample is at least as extreme as a is bounded by

$$Pr[s \succeq an] \leq \left[\left(\frac{p}{a}\right)^a \left(\frac{1-p}{1-a}\right)^{1-a} \right]^n \qquad (1)$$

Table 1 presents probabilities that the support of an itemset in the sample is less than a when its real support is p, for various sample sizes n. For example, given a sample size of 10,000 transactions, the probability that the estimate of a candidate's support is less than 0.8% when its real support is 1% is less than 0.11.

Equation 1 suggests that the sample size should increase as the minimum support decreases. Also, the probability that the estimate is off by more than a certain fraction of the real support depends only on the sample size, not on the database size. Experiments showing the effect of sample size on the running time are given in Section 4.2.

4 Performance Evaluation

In this section, we evaluate the performance of the three algorithms on both synthetic and real-life datasets. First, we describe the synthetic data generation program in Section 4.1. We present some preliminary results comparing the three variants of the stratification algorithm and the effect of changing the sample size in Section 4.2. We then give the performance evaluation of the three algorithms on synthetic data in Section 4.3. We do a reality check of our results on synthetic data by running the algorithms against two real-life data sets in Section 4.4. Finally, we look at the effectiveness of the interest measure in pruning redundant rules in Section 4.5.

We performed our experiments on an IBM RS/6000 250 workstation with 128 MB of main memory running AIX 3.2.5. The data resided in the AIX file system and was stored on a local 2GB SCSI 3.5" drive, with measured sequential throughput of about 2 MB/second.

4.1 Synthetic Data Generation

Our synthetic data generation program is a generalization of the algorithm in [2]; the addition being the incorporation of taxonomies. The various parameters and their default vales are shown in Table 2. We now describe the extensions to the data generation algorithm in more detail.

The essential idea behind the synthetic data generation program in [2] was to first generate a table of potentially frequent itemsets $\mathcal{I}$, and then generate

	$p = 5\%$		$p = 1\%$		$p = 0.5\%$		$p = 0.1\%$	
	a = .8p	a = .9p	a = .8p	a = .9p	a = .8p	a = .9p	a = .8p	a = .9p
$n = 1000$	0.32	0.76	0.80	0.95	0.89	0.97	0.98	0.99
$n = 10,000$	0.00	0.07	0.11	0.59	0.34	0.77	0.80	0.95
$n = 100,000$	0.00	0.00	0.00	0.01	0.00	0.07	0.12	0.60
$n = 1,000,000$	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01

Table 1: Pr[support in sample $< a$], given values for the sample size n, the real support p and a

Parameter		Default Value
$\|\mathcal{D}\|$	Number of transactions	1,000,000
$\|T\|$	Average size of the Transactions	10
$\|I\|$	Average size of the maximal potentially frequent Itemsets	4
$\|\mathcal{I}\|$	Number of maximal potentially Frequent itemsets	10,000
N	Number of items	100,000
R	Number of Roots	250
L	Number of Levels	4-5
F	Fanout	5
D	Depth-ratio $\left(\approx \frac{\text{probability that item in a rule comes from level } i}{\text{probability that item comes from level } i+1}\right)$	1

Table 2: Parameters for Synthetic Data Generation with default values

transactions by picking itemsets from $\mathcal{I}$ and inserting them in the transaction. Details can be found in [2].

To extend this algorithm, we first build a taxonomy over the items.[4] For simplicity, we modeled the taxonomy as a forest rather than a DAG. For any internal node, the number of children is picked from a Poisson distribution with mean μ equal to fanout F. We first assign children to the roots, then to the nodes at depth 2, and so on, till we run out of items. With this algorithm, it is possible for the leaves of the taxonomy to be at two different levels; this allows us to change parameters like the fanout or the number of roots in small increments.

Each item in the taxonomy tree (including non-leaf items) has a weight associated with it, which corresponds to the probability that the item will be picked for a frequent itemset. The weights are distributed such that the weight of an interior node x equals the sum of the weights of all its children divided by the depth-ratio. Thus with a high depth-ratio, items will be picked from the leaves or lower levels of the tree, while with a low depth-ratio, items will be picked from higher up the tree.

Each itemset in $\mathcal{I}$ has a weight associated with it, which corresponds to the probability that this itemset will be picked. This weight is picked from an exponential distribution with unit mean, and then multiplied by the geometric mean of the probabilities of all the items in the itemset. The weights are later normalized so that the sum of the weights for all the itemsets in $\mathcal{I}$ is 1. The next itemset to be put in the transaction is chosen from $\mathcal{I}$ by tossing an $|\mathcal{I}|$-sided weighted coin, where the weight for a side is the probability of picking the associated itemset.

When an itemset X in $\mathcal{I}$ is picked for adding to a transaction, it is first "specialized". For each item $\widehat{x}$ in X which is not a leaf in the taxonomy, we descend the subtree rooted at $\widehat{x}$ till we reach a leaf x, and replace $\widehat{x}$ with x. At each node, we decide what branch to follow by tossing a k-sided weighted coin, where k is the number of children, and the weights correspond to the weights of the children.

See [9] for further details of the candidate generation program.

4.2 Preliminary Experiments

Stratification : Variants The results of comparing the three variants of the stratification algorithm on the default synthetic data are shown in Figure 9. At high minimum support, when there are only a few rules and most of the time is spent scanning the database, the performance of the three variants is nearly identical. At low minimum support, when there are more rules, EstMerge does slightly better than Estimate and significantly better than Stratify. The reason is that even though EstMerge counts a few more candidates than Estimate and Stratify, it makes fewer passes over the database, resulting in better performance.

Although we do not show the performance of Stratify and Estimate in the graphs in Section 4.3, the results were very similar to those in Figure 9. Both Estimate and Stratify always did somewhat worse than EstMerge, with Estimate beating Stratify.

[4] Out of the four parameters R, L, F and N, only three need to be specified, since any three of these determine the fourth parameter.

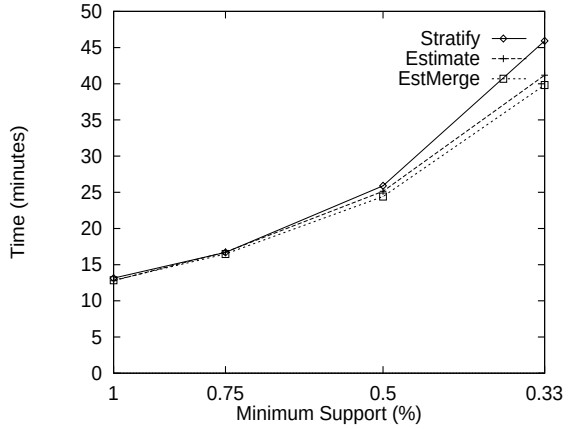

Figure 9: Variants of Stratify

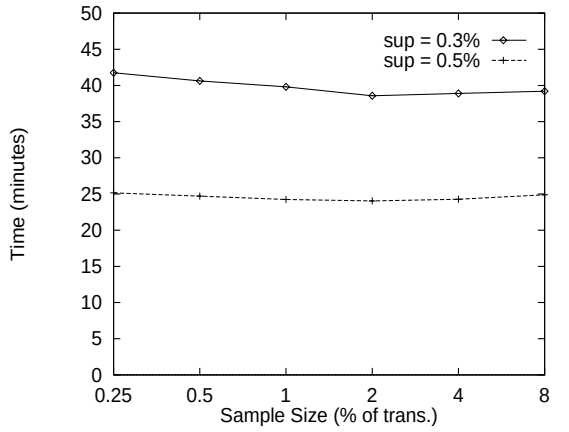

Figure 10: Changing Sample Size

Size of Sample We changed the size of the sample from 0.25% to 8%. The running time was higher at both low sample sizes and high sample sizes. In the former case, the decrease in performance was due to the greater error in estimating which itemsets would have minimum support. In the latter case, it was due to the sampling overhead. Notice that the curve is quite flat around the minimum time at 2%; there is no significant difference in performance if we sample a little less or a little more than 2%.

4.3 Comparison of Basic, Cumulate and EstMerge

We performed 6 experiments on synthetic datasets, changing a different parameter in each experiment. The results are shown in Figure 11. All the parameters except the one being varied were set to their default values. The minimum support was 0.5% (except for the first experiment, which varies minimum support). We obtained similar results at other levels of support, though the gap between the algorithms typically increased as we lowered the support.

Minimum Support: We changed minimum support from 2% to 0.3%. Cumulate and EstMerge were around 3 to 4 times faster than Basic, with the performance gap increasing as the minimum support decreased. At high support, Cumulate and EstMerge took about the same time since there were only a few rules and most of the time was spent scanning the database. At low support, EstMerge was about 20% faster than Cumulate.

Number of Transactions: We varied the number of transactions from 100,000 to 10 million. Rather than showing the elapsed time, the graph shows the elapsed time divided by the number of transactions, normalized such that the time taken by Cumulate for 1 million transactions is 1 unit. Again, EstMerge and Cumulate perform much better than Basic. The ratio of the time taken by EstMerge to the time taken by Cumulate decreases as the number of transactions increases, because when the sample size is a constant percentage, the accuracy of the estimates of the support of the candidates increases as the number of transactions increases.

Fanout: We changed the fanout from 5 to 25. This corresponded to decreasing the number of levels. While EstMerge did about 25% better than Cumulate at fanout 5, the performance advantage deceased as the fanout increased, and the two algorithms did about the same at high fanout. The reason is that at a fanout of 25, the leaves of the taxonomy were either at level 2 or level 3. Hence the percentage of candidates that could be pruned by sampling became very small and EstMerge was not able to count significantly fewer candidates than Cumulate. The performance gap between Basic and the other algorithms decreases somewhat at high fanout since there were fewer rules and a greater fraction of the time was spent just scanning the database.

Number of Roots: We increased the number of roots from 250 to 1000. As shown by the figure, increasing the number of roots has an effect similar to decreasing the minimum support. The reason is that as the number of roots increases, the probability that a specific root would be present in a transaction decreases.

Number of Items/Levels: We varied the number of items from 10,000 to 100,000. The main effect is to change the number of levels in the taxonomy tree, from most of the leaves being at level 3 (with a few at level 4) at 10,000 items to most of the leaves being at level 5 (with a few at level 4) at 100,000 items. Changing the number of items did not significantly affect the performance of Cumulate and EstMerge, but increased

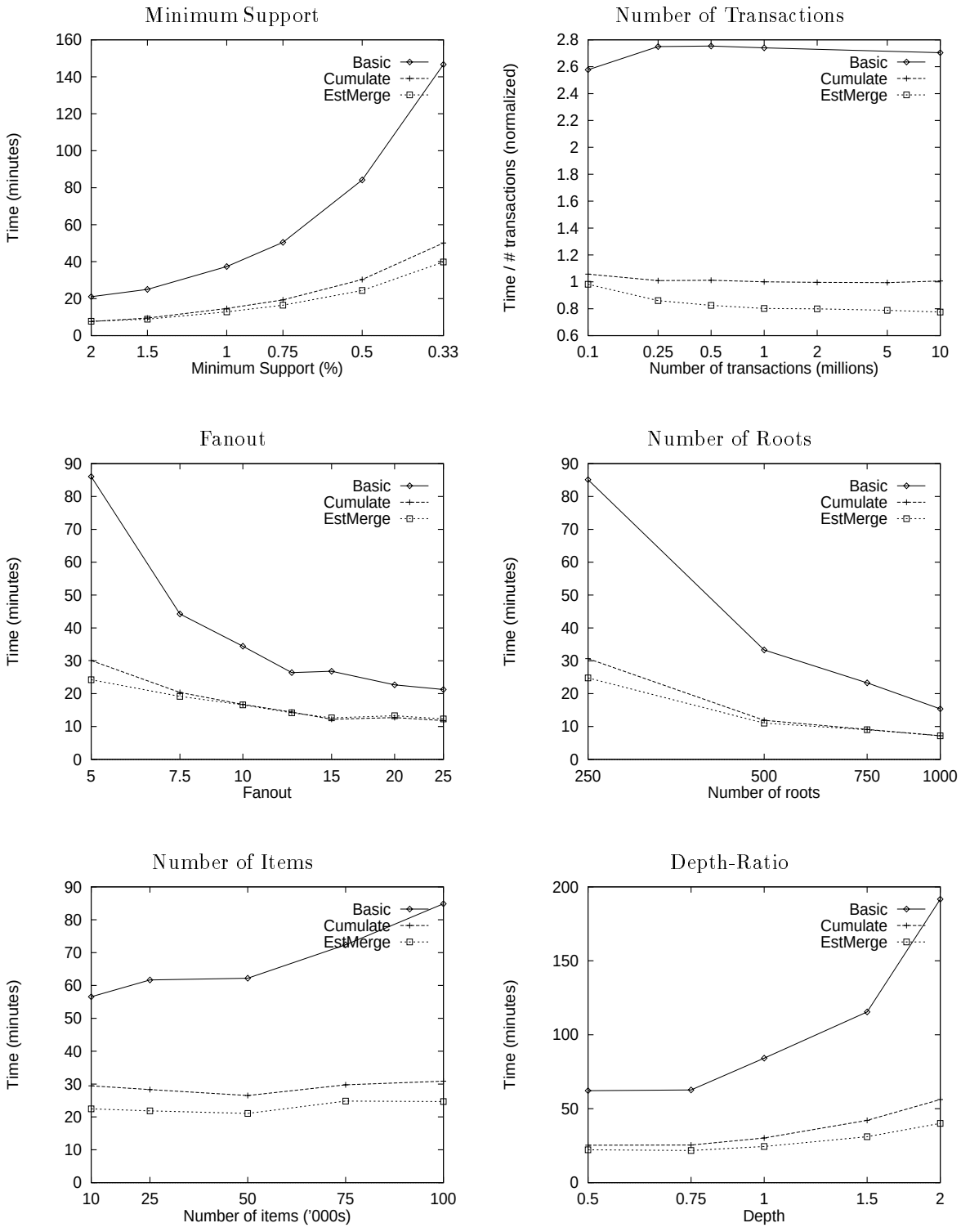

Figure 11: Experiments on Synthetic Data

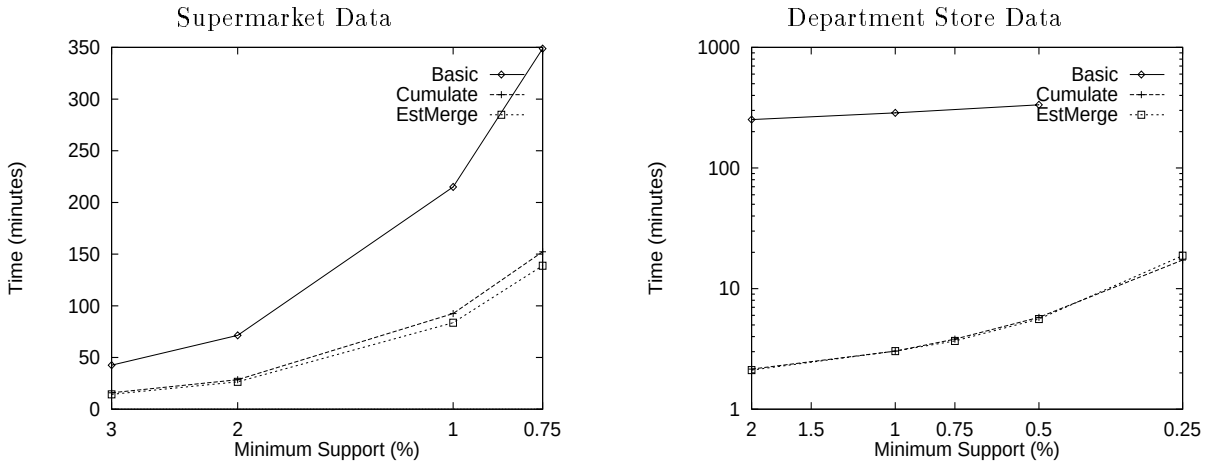

Figure 12: Comparison of algorithms on real data

the time taken by Basic. Since few of the items in frequent itemsets come from the leaves of the taxonomy, the number of frequent itemsets did not change a lot for any of the algorithms. However, Basic had to do more work to find the candidates contained in the transaction since the transaction size (after adding ancestors) increased proportionately with the number of levels. Hence the time taken by Basic increased with the number of items, while the time taken by the other two algorithms remained roughly constant.

Depth-Ratio: We changed the depth-ratio from 0.5 to 2. With high depth-ratios, items in frequent itemsets will tend to be picked from the leaves or lower levels of the tree, while with low depth-ratios, items will be picked from higher up the tree. As shown in the figure, the performance gap between EstMerge and the other two algorithms increased as the depth-ratio increased. At a depth-ratio of 2, EstMerge did about 30% better than Cumulate, and about 5 times better than Basic. The reason is that EstMerge was able to prune a higher percentage of candidates at high depth-ratios.

Summary of Results with Synthetic Data. Cumulate and EstMerge were 2 to 5 times faster than Basic on all the synthetic datasets. EstMerge was 25% to 30% faster than Cumulate on many of the datasets. The advantage decreased at high fanout, since most of the items in the rules came from the top levels of the taxonomy and EstMerge was not able to prune many candidates. There was an increase in the performance gap between Cumulate and EstMerge as the number of transactions increased, since for a constant percentage sample size, the accuracy of the estimates of the support of the candidates increases as the number of transactions increases. Both EstMerge and Cumulate exhibits linear scale-up with the number of transactions.

4.4 Reality Check

To see if our results on synthetic data held in "real world", we ran the algorithms on two real-life datasets.

Supermarket Data This is data about grocery purchases of customers. There are a total of 548,000 items. The taxonomy has 4 levels, with 118 roots. There are around 1.5 million transactions, with an average of 9.6 items per transaction. Figure 12 shows the time taken by the three algorithms as the minimum support is decreased from 3% to 0.75%. These results are similar to those obtained on synthetic data, with EstMerge being a little faster than Cumulate, and both being about 3 times as fast as Basic.

Department Store Data This is data from a department store. There are a total of 228,000 items. The taxonomy has 7 levels, with 89 roots. There are around 570,000 transactions, with an average of 4.4 items per transaction. Figure 12 shows the time taken by the three algorithms as the minimum support is decreased from 2% to 0.25%. The y-axis uses a log scale. Surprisingly, the Basic algorithm was more than 100 times slower than the other two algorithms. Since the taxonomy was very deep, the ratio of the number frequent itemsets that contained both an item and its ancestor to the number of frequent itemsets that did not was very high. In fact, Basic counted around 300 times as many frequent itemsets as the other two algorithms, resulting in very poor performance.

4.5 Effectiveness of Interest Measure

We looked at the effectiveness of the interest measure in pruning rules for the two real-life datasets, at confidence levels of 25% and 50%. For the supermarket data, about 40% of the rules were pruned at a interest

level of 1.1, while about 50% to 55% were pruned for the department store data at the same interest level. In contrast, the interest measure based on statistical significance did not prune any rules at 50% confidence and pruned less than 1% of the rules at 25% confidence (for both datasets). More details about these experiments can be found in [9].

For example, the rule "[Carbonated beverages] and [Crackers] $\Rightarrow$ [Dairy-milk-refrigerated]" was pruned because because its support and confidence were less than 1.1 times the expected support and confidence (respectively) of ancestor "[Carbonated beverages] and [Crackers] $\Rightarrow$ [Milk]", where [Milk] was an ancestor of [Dairy-milk-refrigerated].

5 Summary

We introduced the problem of mining generalized association rules. Given a large database of customer transactions, where each transaction consists of a set of items, and a taxonomy (*is-a* hierarchy) on the items, we find associations between items at any level of the taxonomy. Earlier work on association rules did not consider the presence of taxonomies, and restricted the items in the association rules to the leaf-level items in the taxonomy.

An obvious solution to the problem is to replace each transaction with an "extended transaction" that contains all the items in the original transaction as well as all the ancestors of each item in the original transaction. We could then run any of the earlier algorithms for mining association rules on these extended transactions to get generalized association rules. However, this "Basic" approach is not very fast.

We presented two new algorithms, Cumulate and EstMerge. Empirical evaluation showed that these two algorithms run 2 to 5 times faster than Basic; for one real-life dataset, the performance gap was more than 100 times. Between the two algorithms, EstMerge performs somewhat better than Cumulate, with the performance gap increasing as the size of the database increases. Both EstMerge and Cumulate exhibit linear scale-up with the number of transactions.

A problem users experience in applying association rules to real problems is that many uninteresting or redundant rules are generated along with the interesting rules. We developed a new interest measure that uses the taxonomy information to prune redundant rules. The intuition behind this measure is that if the support and confidence of a rule are close to their expected values based on an ancestor of the rule, the rule can be considered redundant. This measure was able to prune 40% to 60% of the rules on two real-life datasets. In contrast, an interest measure based on statistical significance that did not use taxonomies was not able to prune even 1% of the rules.

Acknowledgment We wish to thank Jeff Naughton for his insightful comments and suggestions.

References

[1] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proc. of the ACM SIGMOD Conference on Management of Data*, pages 207–216, Washington, D.C., May 1993.

[2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. of the VLDB Conference*, Santiago, Chile, September 1994. Expanded version available as IBM Research Report RJ9839, June 1994.

[3] N. Alon and J. H. Spencer. *The Probabilistic Method.* John Wiley Inc., New York, 1992.

[4] T. Hagerup and C. Rüb. A guided tour of Chernoff bounds. *Information Processing Letters*, 33:305–308, 1989/90.

[5] M. Houtsma and A. Swami. Set-oriented mining of association rules. In *Int'l Conference on Data Engineering*, Taipei, Taiwan, March 1995.

[6] H. Mannila, H. Toivonen, and A. I. Verkamo. Efficient algorithms for discovering association rules. In *KDD-94: AAAI Workshop on Knowledge Discovery in Databases*, pages 181–192, Seattle, Washington, July 1994.

[7] J. S. Park, M.-S. Chen, and P. S. Yu. An effective hash based algorithm for mining association rules. In *Proc. of the ACM-SIGMOD Conference on Management of Data*, San Jose, California, May 1995.

[8] G. Piatetsky-Shapiro. Discovery, analysis, and presentation of strong rules. In G. Piatetsky-Shapiro and W. Frawley, editors, *Knowledge Discovery in Databases*, pages 229–248. AAAI/MIT Press, Menlo Park, CA, 1991.

[9] R. Srikant and R. Agrawal. Mining generalized association rules. Research Report RJ 9963, IBM Almaden Research Center, San Jose, California, June 1995.