Efficient Mining of Word Assoictaion Rules from Text Databases

John D. Holt

Dept. of Computer Science and Engineering
Wright State University
Dayton, Ohio 45435, USA

Soon M. Chung †

Dept. of Computer Science and Engineering
Wright State University
Dayton, Ohio 45435, USA

Contact Author: Soon M. Chung
(937)775-5119
(937)775-5133 (Fax)
schung@cs.wright.edu

† This research was supported in part by Ohio Board of Regents, LexisNexis, NCR, and AFRL/Wright Brothers Institute (WBI).

Abstract

In this paper, we propose a new algorithm named Multipass with Inverted Hashing and Pruning (MIHP) for mining association rules between words in text databases. The characteristics of text databases are quite different from those of retail transaction databases, and existing mining algorithms cannot handle text databases efficiently because of the large number of itemsets (i.e., sets of words) that need to be counted. Well-known mining algorithms, such as Apriori [1], Direct Hashing and Pruning (DHP) [12], and FP-growth [8], are evaluated in the context of mining text databases, and are compared with the proposed MIHP algorithm. It has been shown that the MIHP algorithm has much better performance for large text databases. We also evaluated the efficacy of mined association rules between words for measuring the similarity between documents to enhance the text retrieval. In our experiments, for each document relevant to a query, we formed a group of documents having at least one common frequent set of words with the answer document. Then we measured the precision of the documents in the same group as an answer set to the corresponding query. This experiments were performed using the TREC (Text Research Collection) corpus and search results. Our experimental results show that the frequent sets of words mined from our test database are useful in ranking query result sets to improve the precision of retrieval.

Key words: Association rule mining, text retrieval, document similarity, Multipass, Inverted Hashing and Pruning.

1 Introduction

Mining association rules in transaction databases has been demonstrated to be useful and technically feasible in several application areas [2, 3], particularly in retail sales. Let $\mathcal{I} = \{i_1, i_2, \ldots, i_m\}$ be a set of items. Let $\mathcal{D}$ be a set of transactions, where each transaction T contains a set of items. An association rule is an implication of the form $X \Rightarrow Y$, where $X \subset \mathcal{I}$, $Y \subset \mathcal{I}$, and $X \cap Y = \phi$. The association rule $X \Rightarrow Y$ holds in the database $\mathcal{D}$ with *confidence* c if $c\%$ of transactions in $\mathcal{D}$ that contain X also contain Y. The association rule $X \Rightarrow Y$ has *support* s if $s\%$ of transactions in $\mathcal{D}$ contain $X \cup Y$. Mining association rules is to find all association rules that have support and confidence greater than or equal to the user-specified minimum support (called *minsup*) and minimum confidence (called *minconf*), respectively [1]. For example, beer and disposable diapers are items, and $beer \Rightarrow diapers$ is an association rule mined from the database if the co-occurrence rate of beer and disposable diapers (in the same transaction) is not less than *minsup* and the occurrence rate of diapers in the transactions containing beer is not less than *minconf*.

The first step in the discovery of association rules is to find each set of items (called *itemset*) that have co-occurrence rate above the minimum support. An itemset with at least the minimum support is called a *large itemset* or a *frequent itemset*. In this paper, the term *frequent itemset* will be used. The size of an itemset represents the number of items contained in the itemset, and an itemset containing k

items will be called a k-itemset. For example, {beer, diapers} can be a frequent 2-itemset. Finding all frequent itemsets is a very resource consuming task and has received a considerable amount of research effort in recent years. The second step of forming the association rules from the frequent itemsets is straightforward as described in [1]: For every frequent itemset f, find all non-empty subsets of f. For every such subset a, generate a rule of the form $a \Rightarrow (f - a)$ if the ratio of support(f) to support(a) is at least *minconf*.

The association rules mined from point-of-sale (POS) transaction databases can be used to predict the purchase behavior of customers. In the case of text databases, there are several uses of mined association rules between sets of words. They can be used for building a statistical thesaurus. Consider the case that we have an association rule $B \Rightarrow C$, where B and C are words. A search for documents containing C can be expanded by including B. This expansion will allow for finding documents using C that do not contain C as a term. A closely related use is Latent Semantic Indexing, where documents are considered close to each other if they share a sufficient number of associations [7]. Latent Semantic Indexing can be used to retrieve documents that do not have any terms in common with the original text search expression by adding documents to the query result set that are close to the documents in the original query result set. Frequent itemsets mined from a text database may be useful in the task of document ranking.

The word frequency distribution of a text database can be very different from the item frequency distribution of a sales transaction database. Additionally, the number of unique words in a text database is significantly larger than the number of unique items in a transaction database. Finally, the number of unique words in a typical document is much larger than the number of unique items in a transaction. These differences make the existing algorithms, such as Apriori [1], Direct Hashing and Pruning (DHP) [12] and FP-Growth [8], ineffective in mining association rules in the text databases.

A new algorithm suitable for mining association rules in text databases is proposed in this paper. This algorithm is named Multipass with Inverted Hashing and Pruning (MIHP), and is described in Section 4. The results of performance analysis are discussed in Section 5. The new algorithm demonstrated significantly better performance than Apriori, DHP, and FP-Growth algorithms for large text databases.

We evaluated the efficacy of mined association rules between words for measuring the similarity between documents to enhance the text retrieval. In our experiments, for each document relevant to a query, we formed a group of documents having at least one common frequent set of words with the

answer document. Then we measured the precision of the documents in the same group as an answer set to the corresponding query. These experiments were performed using the TREC (Text Research Collection) corpus and search results. Our experimental results show that the frequent sets of words mined from our test database are useful in ranking query result sets to improve the precision of retrieval. The experiment method and the results are discussed in detail in Section 2 and Section 6, respectively.

2 Text Databases

Traditional domains for finding frequent itemsets, and subsequently the association rules, include retail point-of-sale (POS) transaction databases and catalog order databases [2]. The natural item instances are the sales transaction items, but other item instances are possible. For example, individual customer order histories could be used. An item may have a detailed identity, such as a particular brand and size, or may be mapped to a generic identity such as 'bread'. The number of items in a typical POS transaction is well under 100. The mean number of items and distribution varies considerably depending upon the retail operation.

Mining associations rules between words in controlled vocabularies has been done in [4, 5]. Documents labeled with a controlled vocabulary were mined for association rules. Generalized association rules can also be mined when the controlled vocabulary has a thesaurus by substituting broadening terms. The number of controlled vocabulary terms applied to a particular document is much smaller than the number of unique words in the same document. Using controlled vocabularies can improve precision for many kinds of searching. However, when the terms of interest are new, so that they are not in the controlled vocabulary, both precision and recall are adversely affected. Mining association rules to associate free text terms with classification terms has been done in [16]. Documents labeled with classification terms were used as the training set to discover association rules so that new documents could be classified by assigning the appropriate classification terms. The number of candidate itemsets is sharply reduced since every candidate itemset is required to contain at least one classification term, and there are only a small number of classification terms. Our paper is concerned with finding associations between words in text documents without using a controlled vocabulary.

One natural analogue in text data mining to the POS transaction is the document. That is, the words of a document play the same role as the sale items on a POS transaction. Market basket analysis generally ignores the counts for the items purchased. The association of milk and cookies

is made without considering the number of quarts of milk or the number of boxes of cookies on any particular transaction. In a similar manner, text mining is performed without considering the number of times a word appears in the document.

The word distribution characteristics of text data present some scalability challenges to the algorithms that are typically used in mining transaction databases. A random sample of text documents was drawn from the Text Research Collection (TREC) [11]. The sample consisted of the April 1990 Wall Street Journal articles that were in the TREC. There were 3,568 articles and 47,188 unique words. Most of those words occur in only a few of the documents. Some of the key distribution statistics are:

48.2%	of the words occur in only one document;
13.4%	of the words occur in only two documents;
7.0%	of the words occur in only three documents;
4.5%	of the words occur in only four documents;
3.0%	of the words occur in only five documents.

The average number of unique words (including stop-words) in a document was 207, with a standard deviation of 174.2 words. In this sample, only 6.8% of the words occurred in more than 1% of the documents. At a minimum support level of 0.05%, the average number of frequent words per document was 201. The distribution of 207 words in a document on the average is as follows:

6	words are unique to the document;
4	words occur in only one other document;
3	words occur in only two other documents;
2	words occur in only three other documents;
2	words occur in only four other documents;
40	words occur in less than 1% of the documents;
95	words occur in less than 5% of the documents;
126	words occur in less than 10% of the documents;
157	words occur in less than 20% of the documents;
50	words occur in more than 20% of the documents.

When the stop-words are removed from consideration, the average number of unique words in a document falls to 161, with a standard deviation of 140.9 words.

The characteristics of this word distribution have profound implications for the efficiency of association rule mining algorithms. The most important implications are: (1) the large number of items and combinations of items that need to be counted; and (2) the large number of items in each document in the database.

It is commonly recognized in the information retrieval community that words that appear uniformly in a text database have little value in differentiating documents, and further those words occur in a substantial number of documents [13]. It is reasonable to expect that frequent itemsets (i.e., sets of words) composed of highly frequent words (typically considered to be words with occurrence rates above 20%) would also have little value. Therefore, text database miners need to work with itemsets composed of words that are not too frequent, but are frequent enough. The range of minimum and maximum support suitable for word association mining is not known at this time for the general case. We show that for small collections (less than 10,000 documents), a very low minimum support level of 0.06% is required and that a modest maximum support level of about 1% is helpful for increasing precision. It seems clear then that in general, word association mining will require using minimum support levels that are significantly lower than the ones typically used for POS transaction databases.

2.1 Application of Mined Association Rules in Text Retrieval

Frequent itemsets (sets of words) mined from text databases can be used to improve the precision of text retrieval. The TREC corpus described above also contains a set of queries, and for each query there is a list of documents that have been judged to match that query. The April 1990 Wall Street Journal sample contained documents that answered 19 of the 51 TREC-3 queries. The distribution of queries and the number of answer documents for each query for the April 1990 Wall Street Journal articles is:

6 queries	with 1 answer document
4 queries	with 2 answer documents
3 queries	with 3 answer documents
1 query	with 4 answer documents
3 queries	with 5 answer documents
1 query	with 7 answer documents
1 query	with 8 answer documents

Approximately a third of the queries have only one answer document. These 6 queries were not used in our search simulation tests. The inclusion of these 6 queries into the tests would have inflated the precision scores.

The mined frequent itemsets were used in the simulation of the task of using a single matching document as the seed for a query to retrieve additional documents that are similar to the seed document. For each query, each of the matching documents were used as the seed document and document

similarity measures were calculated between the seed document and each document in the collection. In this case, there were 51 seed documents and therefore a simulation of 51 searches.

Document similarity is calculated using the term weights of the frequent words or frequent word sets [13]. The term weight is the Inverse Document Frequency (IDF) of the word or word set. The similarity score of a document with respect to a seed document is then the sum of the weights of the terms that occur both in the document and in the seed document, divided by the sum of the weights of the terms in the seed document.

Then, for each seed document, a ranking list of documents is formed by listing the documents in descending order of their similarity scores with respect to the seed document. The precision of the first matching document in the ranking list is used as the measure to judge the efficacy of using frequent itemsets for the ranking. The precision of the first matching document is simply the inverse of the rank position of the document in the list.

The only criterion used to determine whether a document is matching the query (i.e., relevant to the query) is whether a TREC judge marked the document as a relevant document. The documents retrieved but not considered relevant by the TREC judges were not used for the precision measurement. Queries which returned no document were not considered and queries which returned no relevant document were treated as having a precision of zero.

To form the ranking list of documents for each seed document, first we need to find all the documents having common frequent itemsets with the seed document. To find all those documents, we can use a search query composed of all frequent itemsets contained in the seed document.

Our simulation closely matches the *binary vector space model* of documents [13] that can be used to calculate the closeness between a query and a document. When we use the binary vector space model for information retrieval, the result set of documents are ranked by their similarity scores. Thus, the measures defined above are consistent with the measures that would be obtained if a real information retrieval system were used instead of the simulation.

Instead of using the document as the unit of text mining, we can also use paragraphs and sentences as text units. Documents can also be broken into fixed size fragments. We examined the efficacy of using sentences, paragraphs, 50-word text fragments, 100-word text fragments, and 200-word text fragments as the text units for the mining.

Consider the case of using sentences or paragraphs as the text unit. The frequent itemsets are mined in a manner similar to the mining of frequent itemsets in documents, except that an itemset

is frequent if and only if all of the composing items (i.e., words) occur together in more than *minsup* sentences or paragraphs.

The case of fixed text fragments is different from the cases of using sentences or paragraphs in that overlapping text fragments were used. Each fragment has a 50% overlap with the previous fragment in the same document. Text units made by dividing each document into nonoverlapping fixed size fragments are very arbitrary. Consider two words B and C at positions 41 and 52 in a document. A 50-word fragmentation would place B and C in different fragments, and the relationship between B and C might not be detected. To avoid this problem, a 50% overlap was used between every two consecutive fragments, such that the first 50-word fragment would start at word 1 and end at word 50, the second 50-word fragment would start at word 26 and end at word 75, and so on. This approach does not miss any itemset, but does overstate the support level for low frequency itemsets.

Frequent itemset mining using an alternative text unit instead of using a document required some changes to the matching procedure outlined above. When the document is the text unit, the search query to find all the documents to be included in the rank list of a seed document is simply formed by the set of frequent itemsets found in the seed document. However, when the text unit is smaller than a document, the query is formed by the union of the sets of frequent itemsets found in the text units making up the seed document. The similarity scoring procedure is modified in a similar manner, such that the term weight is represented by the inverse of the frequency of the term in the text units.

3 Existing Algorithms for Mining Association Rules

In this section we describe the Apriori algorithm [1], Direct Hashing and Pruning (DHP) algorithm [12], and FP-Growth algorithm [8] in the context of mining text databases. The performance of the proposed MIHP algorithm is compared with that of these algorithms in Section 5.

The Apriori and DHP algorithms make multiple passes over the database to find all frequent itemsets. In the first pass, the support of individual items are counted and frequent 1-itemsets are determined. Then the frequent 1-items are used to generate the potentially frequent 2-itemsets, called candidate 2-itemsets. In the second pass, we count the support of the candidate 2-itemsets, so that we can determine the frequent 2-itemsets. Frequent 2-itemsets are used to generate the candidate 3-itemsets, and so on. This process is repeated until there is no new frequent itemset. Apriori and DHP are compared with the proposed MIHP algorithm in Section 5.

The FP-Growth algorithm does not generate the candidate itemsets. Instead, it uses a frequent pattern tree (FP-tree) structure to compress the database to avoid repeated scanning of the database.

3.1 Apriori Algorithm

The Apriori algorithm for finding frequent itemsets from a database that consists of transactions is as follows:

1) $F_1 = \{\text{frequent 1-itemsets}\}$;
2) **Comment:** k represents the pass number.
3) **Comment:** F_k is the set of frequent k-itemsets.
4) **for** $(k = 2; F_{k-1} \neq \phi; k++)$ **do begin**
5) **Comment:** C_k is the set of candidate k-itemsets.
6) **Comment:** $F_{k-1} * F_{k-1}$ is the natural join of
F_{k-1} and F_{k-1} on the first $k-2$ items.
7) $C_k = F_{k-1} * F_{k-1}$;
8) **foreach** k-itemset $x \in C_k$ **do**
9) **if** $\exists y \mid y = (k-1)$-subset of x and $y \notin F_{k-1}$
10) **then** remove x from C_k;
11) **foreach** transaction $t \in Database$ **do begin**
12) **foreach** k-itemset x in t **do**
13) **if** $x \in C_k$ **then** $x.count++$;
14) **end**
15) $F_k = \{x \in C_k \mid x.count/|Database| \geq minsup\}$;
16) **end**
17) $Answer = \cup_k F_k$;

The formation of the set of candidate itemsets can be done effectively when the items in each itemset are stored in a lexical order, and itemsets are also lexically ordered. As specified in line 7, candidate k-itemsets, for $k \geq 2$, are obtained by performing the natural join operation $F_{k-1} * F_{k-1}$ on the first $k-2$ items of the $(k-1)$-itemsets in F_{k-1} assuming that the items are lexically ordered in each itemset [1]. For example, if F_2 includes {A, B} and {A, C}, then {A, B, C} is a potential

candidate 3-itemset. Then the potential candidate k-itemsets are pruned in lines 8–10 by using the property that all the $(k-1)$-subsets of a frequent k-itemset should be frequent $(k-1)$-itemsets. This property is *subset closure* property of the frequent itemset. Thus, for {A, B, C} to be a candidate 3-itemset, {B, C} also should be a frequent 2-itemset. This subset-infrequency based pruning step prevents many potential candidate k-itemsets from being counted in each pass k for finding frequent k-itemsets, and results in a major reduction in memory consumption. To count the occurrences of the candidate itemsets efficiently as the transactions are scanned, they can be stored in a hash tree, where the hash value of each item occupies a level in the tree [1, 12].

Apriori does not perform well with a large number of frequent items, because it may generate a considerable number of candidate itemsets that do not have the minimum support. In the collection of April 1990 Wall Street Journal articles, there are approximately 15,000 words out of total about 47,000 words that occur in more than 0.1% of the documents. With Apriori, approximately 112 million candidate 2-itemsets would be generated.

3.2 Direct Hashing and Pruning (DHP) Algorithm

In the Direct Hashing and Pruning (DHP) algorithm, a hashing technique is used to filter out candidate itemsets generated [12]. Each $(k+1)$-itemset in transactions is hashed to a hash value while the occurrences of the candidate k-itemsets are counted by scanning the transactions. If the support count of a hash value is less than the minimum support count, then none of the $(k+1)$-itemsets with that hash value will be included in the set of candidate $(k+1)$-itemsets in the next pass. Pruning candidate itemsets based upon the support counts of their hash values is safe because there may be false positives (i.e., the retained candidate itemsets that are not actually frequent) but there will be no false negatives.

Transaction trimming and transaction pruning methods are also proposed together with DHP [12], so that the size of database to be scanned to count the occurrences of candidate itemsets can be reduced at each pass. Transaction trimming and pruning are based on the subset closure property of frequent itemsets; that is, any subset of a frequent itemset must be a frequent itemset by itself. This property suggests that a transaction may have a candidate $(k+1)$-itemset only if it contains $k+1$ candidate k-itemsets obtained in the previous pass. Thus, as a transaction is scanned to count the occurrences of the candidate k-itemsets, we can determine if this transaction can be pruned from the database in the next pass. On the other hand, if a transaction contains a frequent $(k+1)$-itemset, any item contained in this $(k+1)$-itemset should appear in at least k of the candidate k-itemsets contained

in this transaction. Thus, by counting how many times each item in a transaction is involved in the candidate k-itemsets in that transaction, we can decide whether the item can be eliminated from the transaction in the next pass. A transaction is trimmed by rewriting it without the items that will not contribute to forming the frequent itemsets in the next pass.

How much the DHP can reduce the number of candidate itemsets depends on the number of false positives. The false positives are generated when the hash values are identical for a group of candidate itemsets (the hash synonyms) whose individual frequency is less than the minimum support, but whose hash value frequency is not less than the threshold. The number of candidate itemsets that have the same hash value is directly related to the size of the hash table. Unfortunately, this table is in competition for memory space with the hash tree used to store the candidate itemsets and their counts.

The DHP algorithm is not effective for finding the frequent itemsets in the April 1990 Wall Street Journal articles (see Section 2 for details on the collection) when the required minimum support level is as low as 0.1%. The reason is that there are 47,000 unique words, so that the maximum number of 2-itemsets to be hashed is about 1.1 billion. This is because the hashing of the every 2-itemsets within each document is performed before the pruning of the items. With a minimum support count of four occurrences (i.e., minimum support of 0.1%), if every 2-itemset actually occurred in a document, the hash table would have to accommodate more than 250 million entries to avoid counting every 2-itemset, because the expected number of itemsets with the same hash value would be slightly more than four. For this collection of text data, a hash table of 10 million entries resulted in no pruning of candidate 2-itemsets when the minimum support is 0.1%. There was not enough memory available to try a significantly larger hash table.

3.3 FP-Growth Algorithm

The FP-Growth algorithm starts with a scan of the entire database to determine the frequent items. Then, the database is read again. Within each transaction, infrequent items are removed and frequent items are ordered in descending order of their occurrence counts. The transactions are then placed into a prefix tree named frequent pattern tree (FP-tree). The occurrence counts of itemsets are updated during insertion. Consider the transactions {A, B, C} and {A, B, D} as the first two transactions in the database. The common prefix {A, B} would have an occurrence count of 2. Each of the two suffix branches would have a count of 1. If the third transaction were {A, B, C, D}, the common prefix {A, B} would have an occurrence count of 3, the {C} branch would have an occurrence count of 2, and the

{D} suffix of the {C} branch would have an occurrence count of 1.

There is also a node list for each frequent item linking the nodes of the item into a list. This node list is updated during the construction of the FP-tree.

The mining of the FP-tree can be performed by traversing the node lists of frequent itemsets, starting from the least frequent item and proceeding upward to the most frequent item. Let i be the item under consideration. By following the node list of i, we can find every prefix path for node i. Since all the occurrence counts are available on each of the paths, we can discover the frequent itemsets containing i.

The advantages of FP-Growth come from directly reducing the the database into a more compact FP-tree, then mining the FP-tree in memory without generating candidate itemsets. However, even with the substantial compression provided by the FP-tree, some FP-tree may not fit in main memory, and then the processing time will increase considerably. They proposed a method of efficiently partitioning the database such that each transaction belongs to one and only one partition. The partitioning is done by the common suffix frequent item of the transactions, and the partitions are processed in certain order [8].

There are several aspects of text database that work against the strengths of this approach. First, a typical text document has many words. For example, the average number of frequent items per document (with a minimum support of 0.05%) is 161 in our test database of April 1990 Wall Street Journal articles. There is a total population of about 25,400 frequent words when the stop-word list is used.

Second, only 8 of these words occur in more than 20% of the documents, and about 30 words occur in 10% or more of the documents. A large number of unique prefix paths will sharply limit the degree of compression that can be achieved with the FP-tree.

Finally, the proposed partitioning scheme will results in many very small partitions. For example, there are 6,323 words that occur in just 2 documents (with a minimum support level of 0.05%) out of 3,568 documents. This situation will result in at least 1,784 partitions, because there will not be any common suffix that can exist in more than 2 documents. Considering the large number of items that can be a suffix, it is likely that the number of partitions will be higher. It is important to note though that the number of partitions can not exceed the number of documents.

We implemented the FP-Growth algorithm and evaluated its performance for both transaction databases and text databases. We found it is not scalable when there are many candidate itemsets as

the database size is large and the minimum support level is low.

3.4 Motivation for New Mining Algorithms

Each of the existing algorithms discussed above was shown to be unsuitable for the task of mining frequent itemsets from text databases. The unsuitability was a consequence of not being capable of handling the large number of highly disparate potential frequent itemsets in an effective manner.

The requirement of mining frequent itemsets from a database with a large number of disparate candidate itemsets motivates the development of new mining algorithms.

4 Multipass with Inverted Hashing and Pruning (MIHP) Algorithm

The new Multipass with Inverted Hashing and Pruning (MIHP) algorithm is based on the Multipass-Apriori (M-Apriori) algorithm [9] and the Inverted Hashing and Pruning (IHP) algorithm [10] that we proposed. They are described below in Section 4.1 and Section 4.2, respectively.

Due to M-Apriori, we can reduce the the required memory space by partitioning the frequent 1-itemsets and processing each partition separately. At the same time, by using IHP we can prune many of the candidate itemsets generated for each pass efficiently.

4.1 Multipass-Apriori (M-Apriori) Algorithm

The Multipass-Apriori (M-Apriori) algorithm for mining association rules is based on the Apriori algorithm discussed in Section 3.1. Apriori is not suitable for mining frequent itemsets in text databases because of the high memory space requirement for counting the occurrences of large numbers of candidate itemsets. The Multipass approach partitions the frequent items, and thus partition the candidate itemsets. The partition size is selected to be small enough to fit in the available memory of the system. Each partition is then processed separately. Each partition of items contains a fraction of the set of all items in the database, so that the memory space required for counting the occurrences of the sets of items within a partition will be much less than the case of counting the occurrences of the sets of all the items in the database.

The M-Apriori algorithm is described as follows:

1. Count the occurrences of each item in the database to find the frequent 1-itemsets.

2. Partition the frequent 1-itemsets into p partitions, $P_1, P_2, \ldots, P_p$.

3. Use Apriori algorithm to find all the frequent itemsets whose member items are in each partition, in the order of $P_p, P_{p-1}, \ldots, P_1$, by scanning the database. When partition P_p is processed, we can find all the frequent itemsets whose member items are in P_p. When the next partition P_{p-1} is processed, we can find all the frequent itemsets whose member items are in P_{p-1} and P_p. This is because, when P_{p-1} is processed, the items in P_{p-1} are extended with the frequent itemsets we found from P_p and then counted. This procedure is continued until P_1 is processed.

Assume, without loss of generality, that the frequent 1-itemsets (or, simply items) are ordered lexically. The frequent items are partitioned into p partitions, $P_1, P_2, \ldots, P_p$, such that for every $i < j$, every item $a \in P_i$ is less than every item $b \in P_j$. Thus, the itemsets under consideration for some partition P_i have a particular range of item prefixes. Notice that if the partitions have the same number of items, the potential number of itemsets that will be formed by extending a lexically lower ordered partition will be larger than the potential number of itemsets from a lexically higher ordered partition.

Since the frequent items are ordered lexically, it is important to process the partitions in sequence from the highest ordered one to the lowest ordered one. This processing order is required to support the subset-infrequency based pruning of candidate itemsets. Figure 1 shows an example of partitions, where items 0, 1, 2, 3, 4, and 5 are frequent 1-itemsets and are partitioned into P_1, P_2, and P_3.

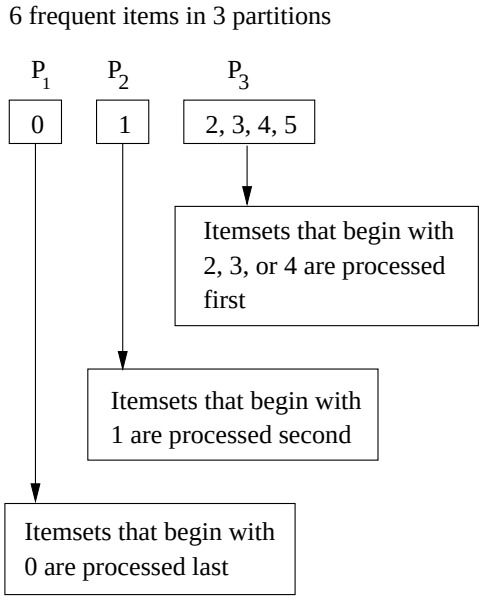

Figure 1: Partitioning a set of 6 frequent items for M-Apriori

When P_3 is being processed, we consider only the candidate itemsets whose members are in

$\{2, 3, 4, 5\}$, not including 0 and 1. Thus, the number of candidates will be smaller compared to the case of considering all the frequent 1-itemsets. Similarly, when P_2 is processed, item 0 is not considered, and many of the candidate itemsets can be pruned. Therefore, it requires less memory space during the processing as one partition is processed at a time.

Suppose that $\{2, 3\}$ is the only frequent 2-itemset found as a result of processing the frequent items in partition P_3. When the next partition P_2 is being processed, item 1 in P_2 is extended with each of the items in P_3 first. As a result, we can find some frequent 2-itemsets including item 1. Let's assume that $\{1, 2\}$, $\{1, 3\}$, and $\{1, 5\}$ are found frequent. Then, $\{1, 2\}$ and $\{1, 3\}$ are joined into $\{1, 2, 3\}$, and we need to check if $\{2, 3\}$ is also frequent to determine whether $\{1, 2, 3\}$ is a candidate 3-itemset or not. Since $\{2, 3\}$ was found frequent when P_3 was processed, $\{1, 2, 3\}$ becomes a candidate 3-itemset. This explains why we need to process the last partition of frequent items first. On the other hand, $\{1, 2, 5\}$ and $\{1, 3, 5\}$ are not candidate 3-itemsets because $\{2, 5\}$ and $\{3, 5\}$ were not found frequent.

In practice, if the estimated number of candidate itemsets to be generated is small after processing a certain number of partitions, then we can merge the remaining partitions into a single partition so that the number of database scans will be reduced.

4.2 Inverted Hashing and Pruning (IHP) Algorithm

Inverted Hashing and Pruning (IHP) is analogous to the Direct Hashing and Pruning (DHP) [12] in the sense that both use hash tables to prune some of the candidate itemsets. In DHP, during the kth pass on the database, every $(k + 1)$-itemset within each transaction is hashed into a hash table, and if the count of the hash value of a $(k + 1)$-itemset is less than the minimum support, it is not considered as a candidate in the next pass. In IHP, for each item, the transaction identifiers (TIDs) of the transactions that contain the item are hashed to a hash table associated with the item, which is named TID Hash Table (THT) of the item. When an item occurs in a transaction, the TID of this transaction is hashed to an entry of the THT of the item, and the entry stores the number of transactions whose TIDs are hashed to that entry. Thus, the THT of each item can be generated as we count the occurrences of each item during the first pass on the database. After the first pass, we can remove the THTs of the items which are not contained in the set of frequent 1-itemsets, and the THTs of the frequent 1-itemsets can be used to prune some of the candidate 2-itemsets. In general, after each pass $k \geq 1$, we can remove the THT of each item that is not a member of any frequent k-itemset, and the remaining THTs can prune some of the candidate $(k + 1)$-itemsets.

Consider a transaction database with seven items: A, B, C, D, E, F, and G. Figure 2 shows the THTs of these items at the end of the first pass. In our example, each THT has five entries for illustration purpose. Here we can see that item D occurred in five transactions. There were two TIDs hashed to 0, one TID hashed to 1, and two TIDs hashed to 4.

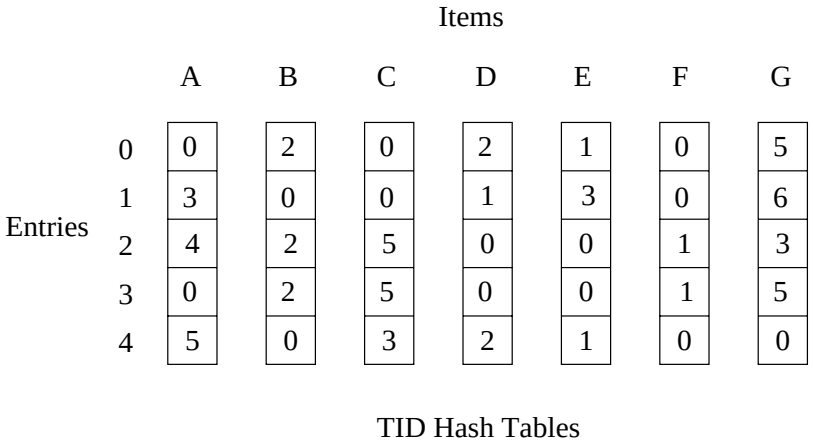

Figure 2: TID Hash Tables at the end of the first pass

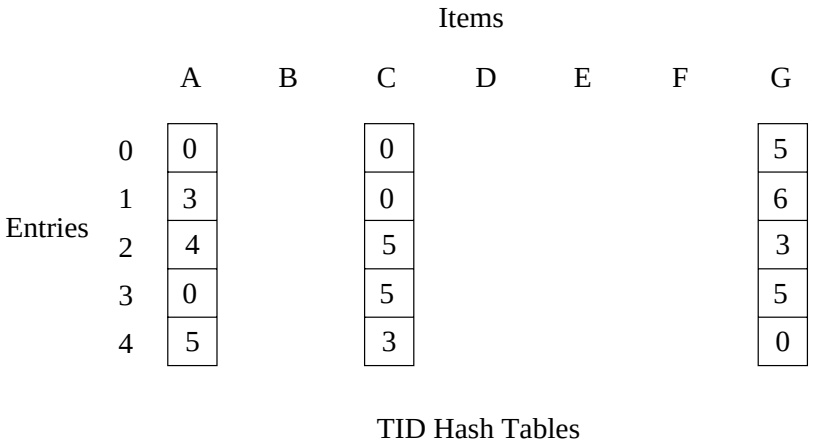

Figure 3: TID Hash Tables of frequent 1-itemsets

If the minimum support count is 7, we can remove the THTs of the items B, D, E, and F as shown in Figure 3. Only the items A, C, and G are frequent and are used to determine C_2, the set of candidate 2-itemsets. As in the Apriori algorithm, {A, C}, {A, G}, and {C, G} are generated as candidate 2-itemsets by pairing the frequent 1-itemsets. However, in IHP, we can eliminate {A, G} from consideration by using the THTs of A and G. Item A occurs in 12 transactions, and item G occurs in 19 transactions. However, according to their THTs, they can co-occur in at most 6 transactions.

Item G occurs in 5 transactions whose TIDs are hashed to 0, and item A occurs in no transactions that have such TIDs. Thus, none of those 5 transactions that contain G also contains A. Item A occurs in 3 transactions whose TIDs are hashed to 1, and item G occurs in 6 transactions with those TIDs. So, in the set of transactions whose TIDs are hashed to 1, items A and G can co-occur at most 3 times. The other THT entries corresponding to the TID hash values of 2, 3, and 4 can be examined similarly, and we can determine items A and G can co-occur in at most 6 transactions, which is below the minimum support level.

In general, for a candidate k-itemset, we can estimate its maximum possible support count by adding the minimum TID hash count of the k items at each entry of their THTs. If the maximum possible support count is less than the required minimum support, it is pruned from the candidate set.

4.3 MIHP Algorithm

The MIHP algorithm is a combination of the Multipass-Apriori algorithm and the Inverted Hashing and Pruning described above, and the details of MIHP is presented below:

1) $Database$ = set of transactions;
2) $Items$ = set of items;
3) transaction = $\langle TID,\ \{x \mid x \in Items\}\rangle$;
4) **Comment:** Read the transactions and count the
   occurrences of each item and create a
   TID Hash Table (THT) for each item
   using a hash function h.
5) **foreach** transaction $t \in Database$ **do begin**
6) **foreach** item x in t **do begin**
7) $x.count++$;
8) $x.THT[(h(t.TID)]++$;
9) **end**
10) **end**
11) **Comment:** F_1 is a set of frequent 1-itemsets.
12) $F_1 = \{x \in Items \mid x.count/|Database| \geq minsup\}$;
13) Partition F_1 into p partitions, $P_1, P_2, \ldots, P_p$;
14) **Comment:** Process the partitions in the order of $P_p, P_{p-1}, \ldots, P_1$.

15) **for** $(m = p; m > 0; m - -)$ **do begin**

16) **Comment:** Find F_k, the set of frequent k-itemsets,
$k \geq 2$, whose members are in partitions $P_m, P_{m+1}, \ldots, P_p$.

17) **for** $(k = 2; F_{k-1} \neq \phi; k + +)$ **do begin**

18) **Comment:** Initialize F_k before the partition P_p is processed.

19) **if** $m = p$ **then** $F_k = \phi$;

20) **Comment:** C_k is the set of candidate k-itemsets whose members
are in $P_m, P_{m+1}, \ldots, P_p$.

21) **Comment:** $F_{k-1} * F_{k-1}$ is the natural join of
F_{k-1} and F_{k-1} on the first $k - 2$ items.

22) $C_k = F_{k-1} * F_{k-1}$;

23) **foreach** k-itemset $x \in C_k$ **do**

24) **if** $\exists y \mid y = (k-1)$-subset of x and $y \notin F_{k-1}$

25) **then** remove x from C_k;

26) **Comment:** Prune the candidate k-itemsets using the THTs.

27) **foreach** k-itemset $x \in C_k$ **do**

28) if $GetMaxPossibleCount(x)/|Database| < minsup$

29) **then** remove x from C_k;

30) **Comment:** Scan the transactions to count the occurrences
of candidate k-itemsets.

31) **foreach** transaction $t \in Database$ **do begin**

32) **foreach** k-itemset x in t **do**

33) **if** $x \in C_k$ **then** $x.count + +$;

34) **end**

35) $F_k = F_k \cup \{x \in C_k \mid x.count/|Database| \geq minsup\}$;

36) **end**

37) **end**

38) $Answer = \cup_k F_k$;

Here, $GetMaxPossibleCount(x)$ returns the maximum number of transactions that may contain a k-itemset x by using the THTs of the k items in x. Let's denote the k items in x by $x[1], x[2], \ldots, x[k]$. Then $GetMaxPossibleCount(x)$ can be defined as follows:

$GetMaxPossibleCount$(itemset x)
begin
  $k = size(x)$;
  MaxPossibleCount = 0;
  for $(j = 0; j < size(THT); j++)$ **do**
    MaxPossibleCount += $min(x[1].THT[j], x[2].THT[j], \ldots, x[k].THT[j])$;
  return (MaxPossibleCount);
end

$size(x)$ represents the number of items in the itemset x and $size(THT)$ represents the number of entries in the THT.

The Partition algorithm [14] used the list of TIDs of the transactions containing each item, called a TID-list. Once the TID-list is generated for each item by scanning the database once, we don't need to scan the database again, because the support count of any candidate itemset can be obtained by intersecting the TID-lists of the individual items in the candidate itemset. However, a major problem is that the size of the TID-lists would be too large to maintain when the number of transactions in the database is very large. On the other hand, the number of entries in the THTs can be limited to a certain value, and it has been shown that even a small number of entries can effectively prune many candidate itemsets.

4.3.1 Additional Efficiency for MIHP

For further performance improvement, the MIHP algorithm can be used together with the transaction trimming and pruning method proposed as a part of the DHP algorithm [12]. The concept of the transaction trimming and pruning is as follows: During the kth pass on the database, if an item is not a member of at least k candidate k-itemsets within a transaction, it can be removed from the transaction for the next pass (transaction trimming), because it cannot be a member of a candidate $(k+1)$-itemset. Moreover, if a transaction doesn't have at least $k+1$ candidate k-itemsets, it can be

removed from the database (transaction pruning), because it cannot have a candidate $(k+1)$-itemset. It is convenient to use a slightly weaker form of the rule for MIHP. The transaction trimming rule as stated is only applied to the items that are in the F_1 partition being processed. The items that are not in the partition being processed are removed if they are not members of any candidate itemset during the current pass. Also, a transaction is pruned during the kth pass if it does not have at least k candidate k-itemsets, each of which contains one or more items from the current F_1 partition being processed.

Transaction trimming and pruning is synergistic with the candidate pruning by IHP. The reduction of candidate k-itemsets for the kth pass, $k \geq 2$, will result in additional items to be trimmed during the kth pass. In addition, the partitioning of the frequent 1-itemsets based on the Multipass is synergistic with the transaction trimming and pruning because more items and transactions can be removed as one partition is processed at a time.

5 Performance Analysis of MIHP

Some performance tests have been done with MIHP. The first objective was to assess whether the Multipass approach combined with Inverted Hashing and Pruning (IHP) would improve the performance. The second objective was to assess the scalability of the MIHP miner. To meet these objectives, we studied the performance of five miners: Apriori, DHP, M-Apriori, IHP, and MIHP. All these miners were derived from the same code base. All of the test runs were made on a machine with a 400 MHz Pentium processor and 512 Mbytes of memory. All miners were written in Java, and the IBM JVM 1.1.8 was used.

For all of the tests, the JVM memory for objects was constrained (via the *mx* parameter) to 456 Mbytes. The initial heap size was set to 456 Mbytes (via the *ms* parameter) to control the effect of heap growth overhead on the performance. The partition size used for the M-Apriori and MIHP was 100 frequent items, and the hash table size for MIHP and IHP was 500 entries. The hash table size for DHP was 500,000 entries. The minimum support ranged from 3% to 1.75% in the runs comparing the miners. For the test of scalability at various document collection sizes, 1.75% minimum support level was used.

The minimum support values for the comparison test runs was selected such that only 456 Mbytes of memory would be required. The rationale for this choice is to study the performance of the algorithms

without the additional complication of accounting for the impact of paging upon the performance. MIHP was the only miner able to run at the 1.75% minimum support level under the 456 Mbytes memory constraint. The memory constraint was lifted to 512 Mbytes, and the Apriori, M-Apriori, and IHP miners were allowed to run for 100,000 seconds each before they were terminated.

We did not use a stop-word list in this set of tests, but instead used a maximum support threshold of 20% to remove the common words. This threshold removed about 150 words from consideration. We also did not perform any stemming of the words. We did however monocase the words.

The comparison runs were made against the April 1990 Wall Street Journal articles. There were 3,568 documents in 11.7 Mbytes, and 47,188 unique words were in the collection. The number of frequent itemsets for each of the minimum support levels can be seen in Figure 4. At the 2% minimum support level, there were total 131,793 frequent itemsets of all sizes. At the 1.75% minimum support level, this jumped to 242,569 frequent itemsets.

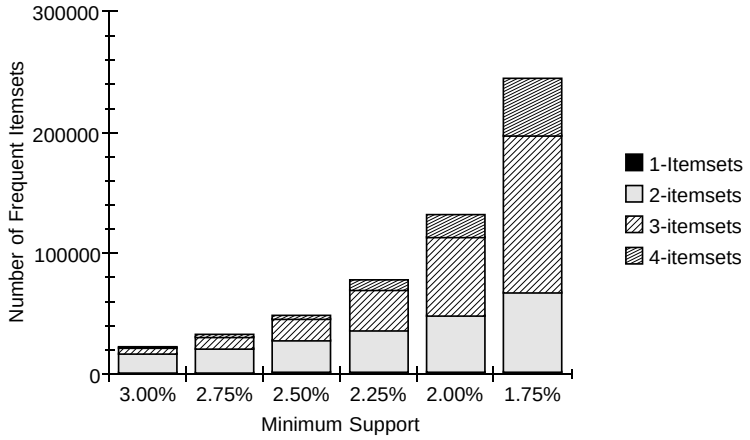

Figure 4: Number of frequent itemsets

In Figure 5, we see that the effect of various support levels upon the execution time of the five miners. There are three distinct performance groups visible. The slowest miner is DHP. The distribution of words in text documents and the large document sizes do not play to the strength of DHP, because the overhead of direct hashing for each pass on the database offsets the benefit of pruning candidate itemsets. A variation of DHP that uses the direct hashing only for the second pass performs about as well as Apriori. The middle group is Apriori and M-Apriori. Note that M-Apriori is slightly faster than Apriori at the lower minimum support levels due to the improvement from the Multipass approach. The faster group is composed of IHP and MIHP. The difference in performance between the two

groups is due to the combination of reducing the number of candidate itemsets and the processing efficiency gained through transaction trimming and pruning. Again, we see at the lower thresholds, the MIHP algorithm is faster than the IHP algorithm. It is important to recall that these runs were memory constrained such that the JVM memory for objects was limited to 512 Mbytes. Since the main memory size was also 512 Mbytes, there was no virtual memory paging affecting the execution times. Only MIHP was able to complete at the 1.75% minimum support level. The Apriori, M-Apriori, and IHP miners were run at the 1.75% minimum support level, but were terminated after running 100,000 seconds.

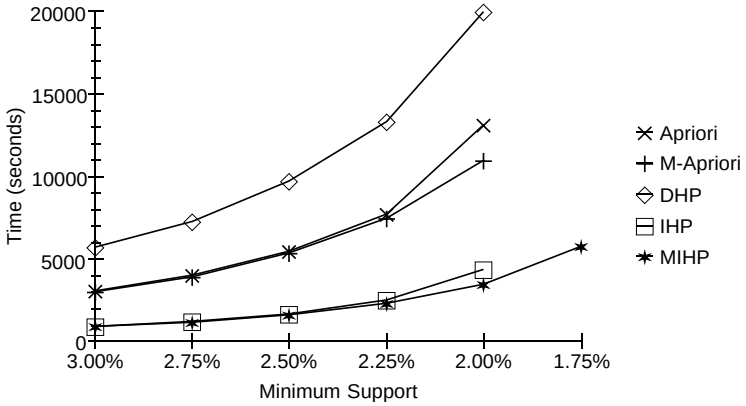

Figure 5: Comparison of execution times (on 3,568 documents)

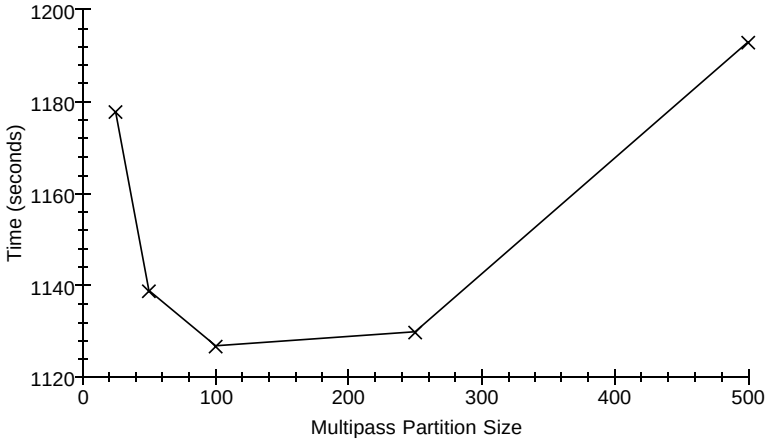

Figure 6: Effect of Multipass partition size in MIHP

For MIHP, the effect of the partition size (of frequent items) was examined in a series of runs with the minimum support level of 2.5% and partition sizes of 25, 50, 100, 250, and 500 frequent items.

In Figure 6 we see that a partition size of about 100 frequent items is appropriate. This implies that Multipass is effective only up to the point where the efficiency of working with a smaller number of candidate itemsets for each pass overcomes the overhead of additional scanning of the database.

The effect of the TID hash table size was examined in a series of runs with a minimum support level of 2.5% and TID hash table sizes of 10, 25, 50, 100, 250, and 500 entries. Figure 7 shows that there is a sharp reduction in the number of candidate 2-itemsets as the number of TID hash table entries increases up to 250, but only a modest decrease after that. In Figure 8 we see that the change in the size of TID hash table, and hence the change in the number of candidate itemsets, has little effect upon the total execution time until it becomes as large as 500 entries. At this point, the effect of candidate pruning clearly offsets the overhead of using the TID hash tables.

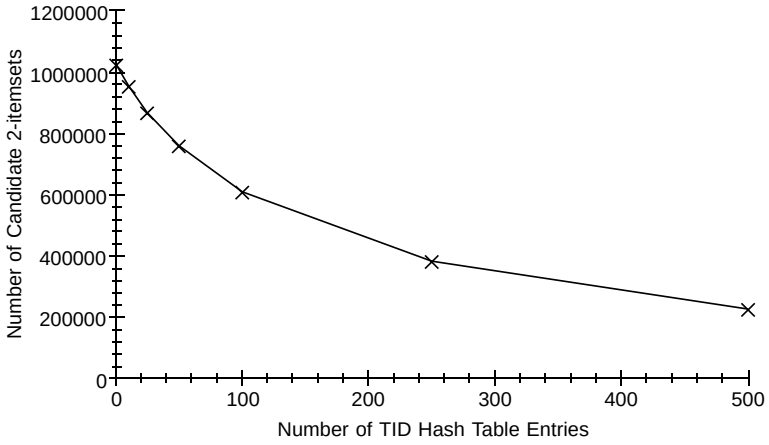

Figure 7: Effect of TID hash table size on the candidates

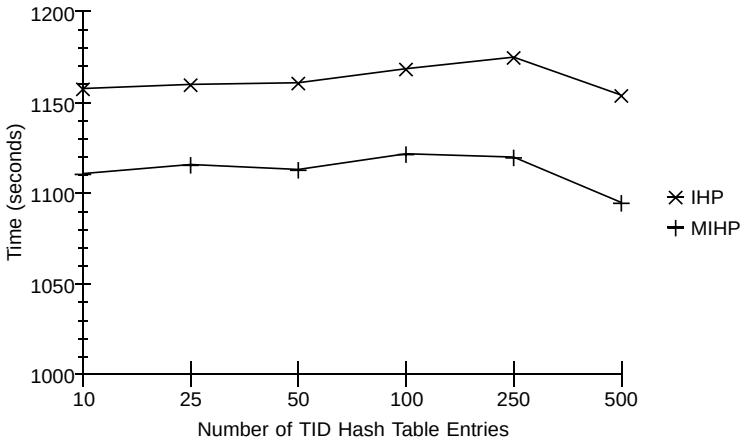

Figure 8: Effect of TID hash table size on the execution time

The above results suggest that the MIHP algorithm is more effective than Apriori, DHP, M-Apriori, and IHP algorithms for mining frequent itemsets from text documents. The next tests were conducted to evaluate how well MIHP scaled in terms of the number of documents to be processed and the number of frequent itemsets mined.

There were four document collections used for the scalability test: a two-week collection (from 3/2/1992 to 3/13/1992) of 1,739 documents; a one-month collection (April 1990) of 3,568 documents; a two-month collection (September and October of 1991) of 7,361 documents; and the largest collection was for three months (January, February, and March in 1992) containing 10,163 documents. The minimum support level was 1.75% for all test runs.

In Figure 9 we see that the time per frequent itemset mined increases proportionally with the number of documents in the collection. This suggests that MIHP will scale linearly with respect to the number of documents processed. In Figure 10 we see that the time per frequent itemset mined decreases as the minimum support level decreases. The time per candidate itemset processed was constant at about 1 msec. The decrease in time per frequent itemset mined is due to the increase in the number of candidate itemsets that become frequent itemsets as the minimum support level decreases.

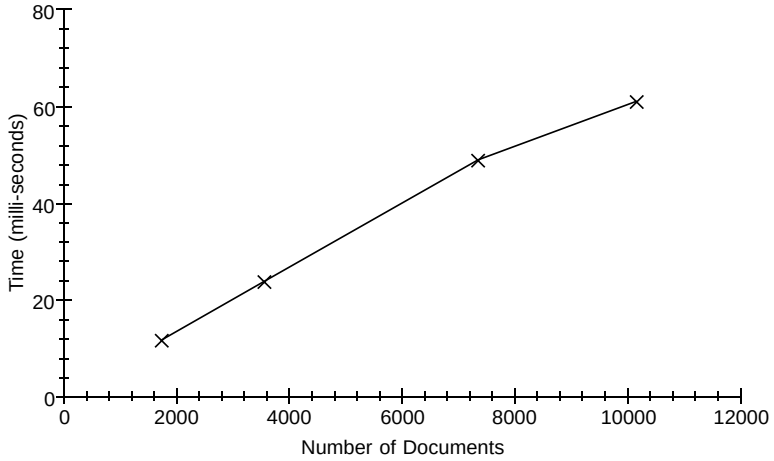

Figure 9: Time per frequent itemset mined (1.75% minsup)

We also implemented the FP-Growth algorithm and run it with the same memory constraint of 512 Mbytes. We found that FP-Growth has slightly better performance than the Apriori-based miners when processing the April 1990 Wall Street Journal articles. It is because FP-Growth does not generate the candidate itemsets, and in that case, the FP-tree is small enough to be traversed quickly to find all the frequent itemsets. We evaluated the FP-Growth algorithm and the MIHP algorithm with a larger

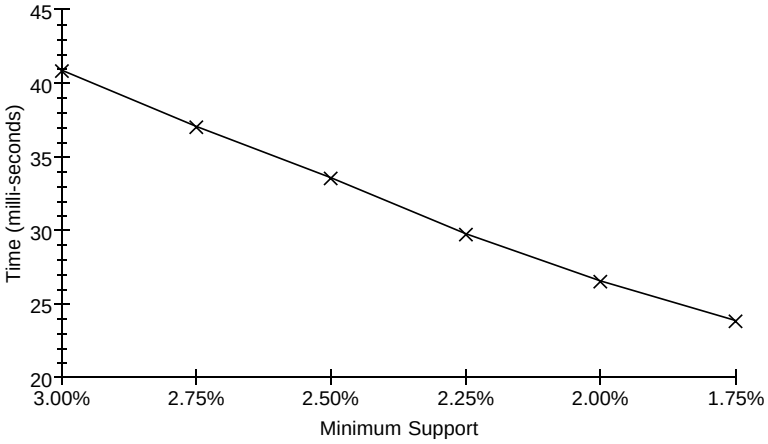

Figure 10: Time per frequent itemset mined (3,568 documents)

database consisted of 21,703 Wall street Journal articles published in the 6 months period from April 1990 to September 1990. The total execution times are shown in Figure 11, and we can confirm that FP-growth is not scalable as we discussed in Section 3.3. As the database size increases, so does the FP-tree, and it takes a lot of time to traverse the large FP-tree.

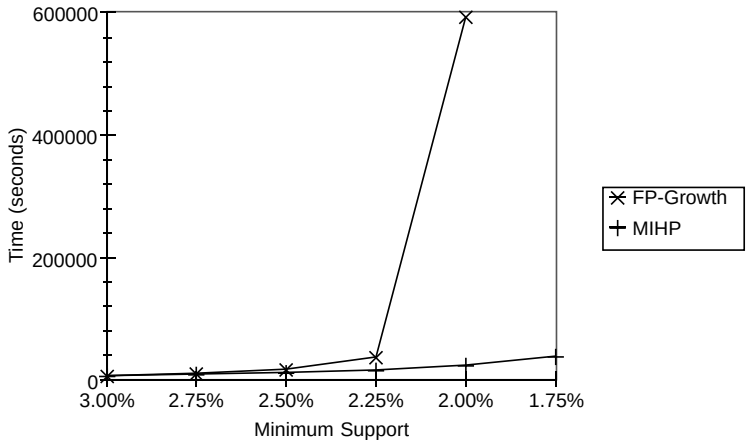

Figure 11: Comparison of execution times (on 21,703 documents)

6 Performance Analysis of Text Retrieval Using Association Rules between Words

The document collection used for the precision experiments was the April 1990 Wall Street Journal articles from the TREC [11]. Section 2 provides the summary statistics for this collection. The tests to measure the effectiveness of frequent 2-word and 3-word sets for text retrieval were conducted at minimum support levels between 1% and 0.06%. The minimum support level of 0.06% corresponds to 2 documents in the collection. Section 2 discusses the measures used. The text retrieval was simulated by scoring the similarity of the documents in the collection with each of the matching documents of queries. In addition to the standard use of a document as a the text unit, we examined several alternative text units. These alternatives included sentences and paragraphs. The sentence and paragraph text units provided the best results. However, sentence and paragraph information is not always readily available. We first present the results of using documents and then present the results from using the smaller alternative text units.

There was no normalization for document size and document term frequencies were not used. These are well known techniques that are used to improve the precision, but were not used in our experiments so that we could focus on the effect of using frequent word sets on the precision.

In the following discussion of the evaluation we will use the general terminology of frequent 1-itemsets, frequent 2-itemsets, and frequent 3-itemsets to refer to frequent single words, frequent 2-word sets, and frequent 3-word sets, respectively.

A standard list of stop-words were used [6], and we also used a maximum support level of 20% to remove too frequent word sets from consideration. At the 1% minimum support level, we found that the 20% maximum support level provides better results than the stop-word list for frequent 3-itemsets. The average precision scores for the 20% maximum support and the stop-word list were 0.091 and 0.083, respectively. Using the 20% maximum support provided approximately a 10% improvement in the average precision. It is interesting to note that the words removed using the maximum support level included content bearing words such as: business, exchange, billion, company, and chairman.

The evaluation showed that frequent 3-itemsets improves the precision considerably. Figure 12 shows the precision scores for frequent 3-itemsets, frequent 2-itemsets, and frequent 1-itemsets. The frequent 3-itemsets and 2-itemsets were mined with the minimum support level of 0.06% and the maximum support level of 1%. The frequent 1-itemsets were mined using the stop-word list and the

minimum support level of 0.06%, which corresponds to 2 documents in the collection.

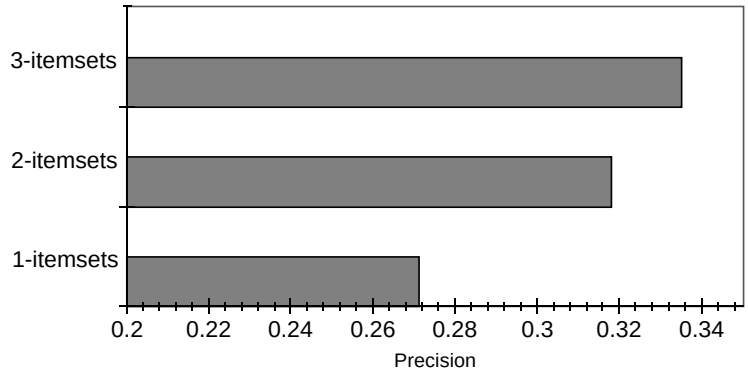

Figure 12: Comparison of different itemset sizes

We also applied the maximum support level for the frequent 1-itemsets, but the average precision declined. The average precision scores for the stop-word list and the 1% maximum support level were 0.291 versus 0.271, which is about 7% decrease. Our conclusion is that using the maximum support level is counterproductive for frequent 1-itemsets, but is productive for frequent 2-itemsets and 3-itemsets.

It is reasonable to believe that the average precision would be enhanced if the items making up the frequent itemsets are highly correlated; i.e., when the association rules derived from the frequent itemsets have high confidence levels. Figure 13 shows that there is supporting evidence for the hypothesis that frequent itemsets whose association rules have high confidence are effective in improving the precision of the first matching document in the ranked list. At the 80% confidence level, the precision of the first document was increased to 0.368, which is a significant improvement to the base case of all of the frequent itemsets.

However, restricting the frequent itemsets to those whose association rules have high confidence will cause a loss in the recall. At the 40% confidence level, 33% of the queries failed to retrieve any document; i.e., no document in the ranking list. The percentage of failed queries increased as the confidence level increased, as shown in Figure 14.

We examined the effect of varying the maximum support level from 1.5% to 0.5%. Figure 15 shows that average precision for 3-itemsets is quite consistent and significantly higher than the average precision obtained from using 1-itemsets (0.291). Figure 16 shows the query failure rate as the maximum support level changes. There are no query failures at the maximum support level of 1%, and one query failure at 0.875%.

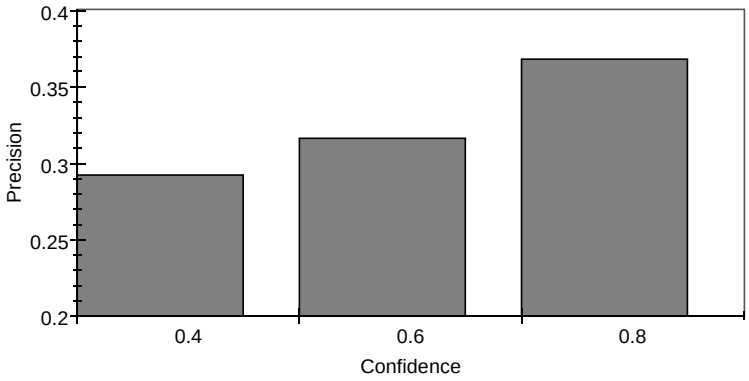

Figure 13: Effect of association rule confidence on the average precision

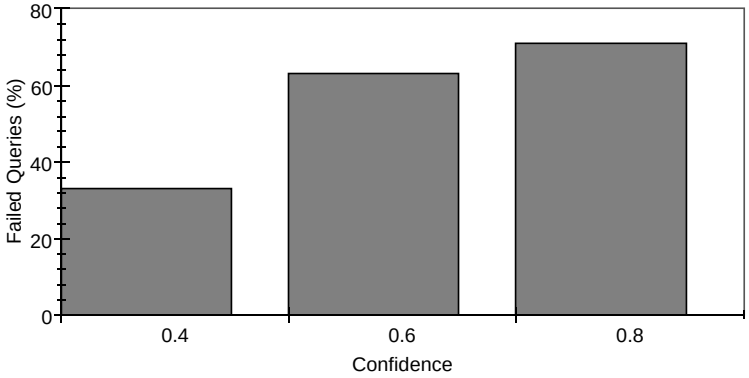

Figure 14: Query failure and association rule confidence

The average precision at the 1% maximum support level was 0.335, which represents an improvement of 15% over the case of using 1-itemsets with no query failure. The average precision at the 0.875% maximum support level was 0.342, which represents an improvement of 18% over the case of using 1-itemsets with only a 2% query failure rate.

We examined several alternative text units. Text fragments with a fixed number of words did not perform well. We tried fragment sizes of 50, 100, and 200 words. The detailed description of the text fragments can be found in Section 2. The precision scores using 3-itemsets ranged from 0.307 for the 50-word fragments to 0.334 for the 200-word fragments. Recall that the best whole document score was 0.305 without confidence level screening. The average number of unique words in a document was 221 words without the stop-word list, and was 161 words with the stop-word list. Thus, it is not surprising that the 200-word fragment precision score would be very close to the whole document precision score.

Sentences and paragraphs are natural text units for mining. However, sentence and paragraph

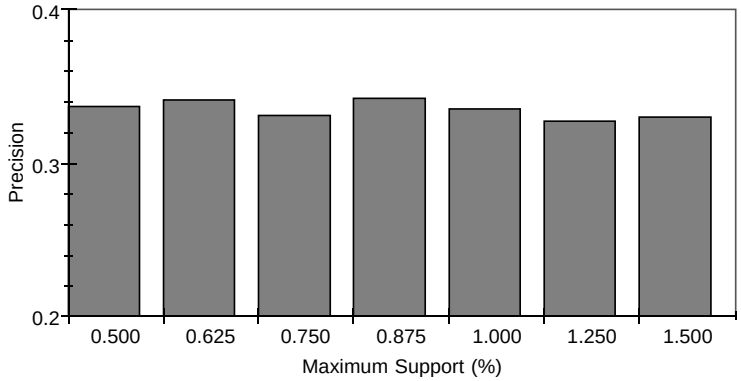

Figure 15: Effect of maximum support on the average precision

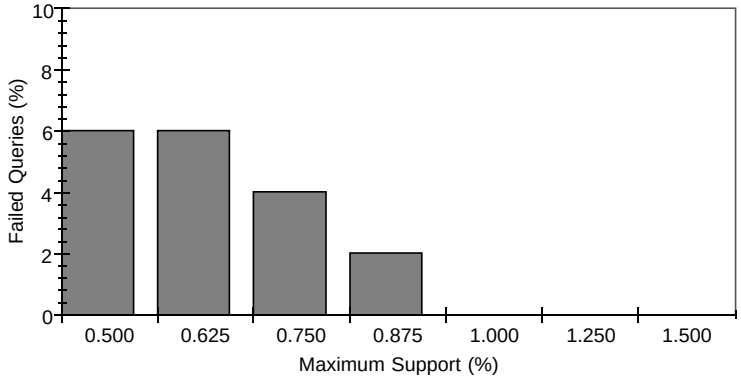

Figure 16: Query failures and maximum support

information is not always available. If the text has been properly annotated with a markup language, sentence and paragraph extraction can be reliably performed. The Wall Street Journal articles in TREC did not have markups for paragraphs or sentences. An indentation style was used for paragraphs, so they could be inferred by the sequence of a new line followed by several space characters. The sentence boundaries were inferred by the punctuation found.

Figure 17 shows the results from using sentences and paragraphs as the text units. The 2-itemsets for sentences and paragraphs were much better than the 3-itemsets. The precision scores from using 2-itemsets in sentences and paragraphs were 0.377 and 0.370, respectively, which is an improvement of about 12.5% over using 3-itemsets from documents.

It is not quite clear why the precision score of 3-itemsets was lower than that of 2-itemsets when sentences and paragraphs were used as the text units instead of the whole document.

We examined the hypothesis that the 3-itemsets retrieved fewer relevant documents than 2-itemsets,

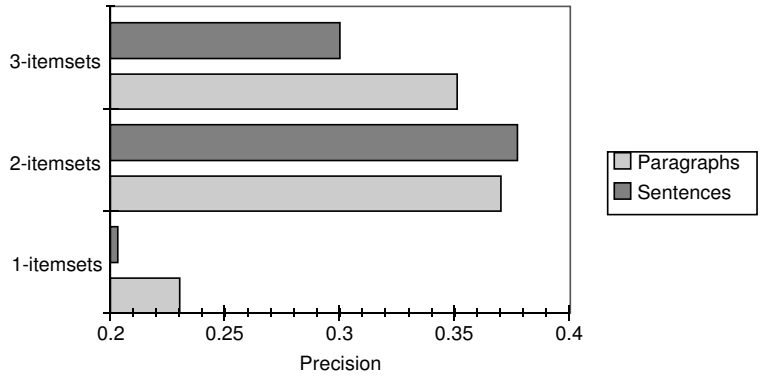

Figure 17: Comparison of different itemset sizes

and therefore there was an increase in the number of queries with zero precision when 3-itemsets were used instead of 2-itemsets. This hypothesis was not well supported by the empirical results.

There was a substantial increase in the number of queries which failed to retrieve relevant documents when 3-itemsets were used instead of 2-itemsets. The queries with no relevant documents when 3-itemsets were used, but retrieved relevant documents when 2-itemsets were used, decreased the average precision by 0.016, which is more than 50% of the decline from 0.370 to 0.351.

Unfortunately, the same phenomenon occurs when the whole document is used as the text unit. In the whole document case, queries that failed to retrieve any relevant documents with 3-itemsets but did retrieve relevant documents with 2-itemsets provided a decrease of 0.032 in precision. However, the average precision increased from 0.318 to 0.335 in the whole document case. The decline in precision caused by retrieving fewer relevant documents was offset by the increase in precision from the use of 3-itemsets.

We are left with the conclusion that frequent 3-itemsets mined by using the whole documents are somehow better for ranking than frequent 3-itemsets mined by using sentences or paragraphs. Our conjecture is that the narrow scope of a paragraph or a sentence does not allow the disparate words in a document to be correlated as much as the broader scope of using the whole document as the unit of text.

Figure 18 shows the effect of varying the maximum support level. The results are similar to those observed when using the whole document as the text unit. Thus, having a maximum support appears to be quite useful in improving the precision. We varied the maximum support level from 0.5% to 2%, and found that a maximum support level of 1% provided the best results.

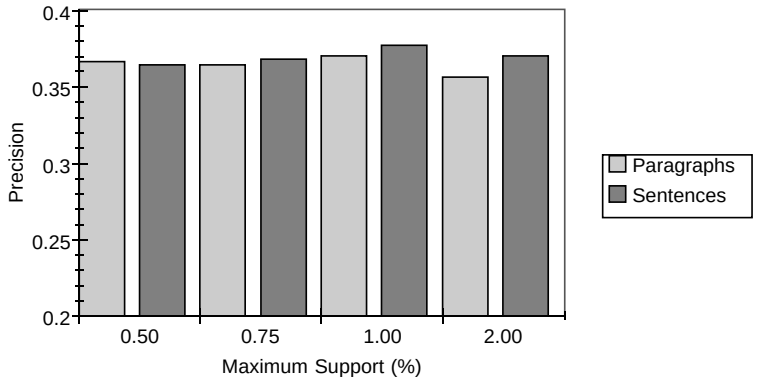

Figure 18: Effect of maximum support on the average precision

The effect of a low maximum support level for frequent itemsets motivates the question of whether the correlation between items was causing the improvement. So, we also examined the case of restricting the itemsets to the subset of frequent itemsets with high confidence level. This was effective with documents, but was ineffective for the sentence and paragraph cases. The decline in the precision scores was substantial.

7 Conclusions

The main conclusions that can be drawn from this study are centered around the nature of the text databases and the use of the mined frequent itemsets. The distribution of words in text document collections and the number of unique words in a document make the problem of finding frequent itemsets (i.e., sets of words) in text databases very different from the case of traditional point-of-sale transaction databases. This difference motivated us to develop the new MIHP (Multipass with Inverted Hashing and Pruning) algorithm for text databases.

Our performance analyses show that the Multipass approach can be effective with text databases. The key performance factors appear to be the reduction in the amount of required memory space and the reduction of the documents during mining. The Multipass approach reduces the number of objects in memory during each pass by partitioning the frequent 1-itemsets, and processes each partition separately. The Multipass approach enhances the effect of transaction trimming and pruning by increasing the number of items pruned during a pass. Moreover, Inverted Hashing and Pruning (IHP) can prune some of the candidate itemsets generated for each pass on the database efficiently. Since many TID hash tables are pruned before we count the occurrences of the candidate 2-itemsets,

most of the memory used for holding the TID hash tables in the first pass on the database is available to hold the candidate 2-itemsets for the second and subsequent passes. In large part, this effect is a result of the distribution of word occurrences discussed in Section 2. The large number of words with very low occurrence rates results in a situation that the preponderance of the TID hash tables generated in the first pass are pruned prior to the initiation of the second pass.

We also show that frequent 3-itemsets (frequent sets of 3 words) can be used to increase the precision of text retrieval. Our results also show that it is helpful to impose a maximum support level for the candidate itemsets so that candidate itemsets with support greater than the maximum are removed.

In addition, we show that frequent itemsets mined from sentences and paragraphs increase the precision of text retrieval, and they provide a better performance than the frequent itemsets mined from documents. Our future research direction is to parallelize the MIHP miner.

References

[1] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules," *Proc. of the 20th VLDB Conf.*, 1994, pp. 487–499.

[2] S. Brin, R. Motwani, J. Ullman, and S. Tsur, "Dynamic Itemset Counting and Implication Rules for Market Basket Data," *Proc. of ACM SIGMOD Int'l Conf. on Management of Data*, 1997, pp. 255–264.

[3] M. S. Chen, J. Han, and P. S. Yu, "Data Mining: An Overview from a Database Perspective," *IEEE Trans. on Knowledge and Data Engineering*, Vol. 8, No. 6, 1996, pp. 866–883.

[4] R. Feldman and H. Hirsh, "Finding Associations in Collections of Text," *Machine Learning and Data Mining: Methods and Applications*, R. Michalski, I. Bratko, and M. Kubat (editors), John Wiley and Sons, 1998, pp. 223-240.

[5] R. Feldman, I. Dagen, and H. Hirsh, "Mining Text Using Keyword Distributions," *Journal of Intelligent Information Systems*, Vol. 10, No. 3, 1998, pp. 281-300.

[6] C. Fox, "Lexical Analysis and Stoplists," *Inforamtion Retrieval: Data Structures and Algorithms*, W. Frakes and R. Baeza-Yates (editors), Prentice Hall, 1992, pp. 102-130.

[7] M. Gordon and S. Dumais, "Using Latent Semantic Indexing for Literature Based Discovery," *Journal of the Amer. Soc. of Info Science*, Vol. 49, No. 8, 1998, pp. 674–685.

[8] J. Han, J. Pei, and Y. Yin, "Mining Frequent Patterns without Candidate Generation," *Proc. of ACM SIGMOD Int'l Conf. on Management of Data*, 2000, pp. 1–12.

[9] J. D. Holt and S. M. Chung, "Multipass Algorithms for Mining Association Rules in Text Databases," *Knowledge and Information Systems*, Vol. 3, No. 2, Springer-Verlag, 2001, pp. 168–183.

[10] J. D. Holt and S. M. Chung, "Mining Association Rules Using Inverted Hashing and Pruning," *Information Processing Letters*, Vol. 83, No. 4, Elsevier Science, 2002, pp. 211–220.

[11] National Institute of Standards and Technology (NIST), *Text Research Collection*, 1997.

[12] J. S. Park, M. S. Chen, and P. S. Yu, "Using a Hash-Based Method with Transaction Trimming for Mining Association Rules," *IEEE Trans. on Knowledge and Data Engineering*, Vol. 9, No. 5, 1997, pp. 813–825.

[13] G. Salton, *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*, Addison-Wesley Publishing, 1988.

[14] A. Savasere, E. Omiecinski, and S. Navathe, "An Efficient Algorithm for Mining Association Rules in Large Databases," *Proc. of the 21st VLDB Conf.*, 1995, pp. 432–444.

[15] H. Toivonen, "Sampling Large Databases for Association Rules," *Proc. of the 22nd VLDB Conf.*, 1996, pp. 134–145.

[16] O. R. Zaiane and M. L. Antoine, "Classifying Text Documents by Associating Terms with Text Categories," *Proc. of the 13th Australian Database Conf.*, 2002.