Mining Association Rules with Adjustable Accuracy

Jong Soo Park*
Dept of Comput. Sci.
Sungshin Women's Univ.
Seoul, Korea
jpark@cs.sungshin.ac.kr

Philip S. Yu
IBM T.J. Watson Res. Ctr.
P.O.Box 704
Yorktown, NY 10598, U.S.A.
psyu@watson.ibm.com

Ming-Syan Chen
Elect. Eng. Department
National Taiwan Univ.
Taipei, Taiwan, ROC
mschen@cc.ee.ntu.edu.tw

Abstract

In this paper, we devise efficient algorithms for mining association rules with adjustable accuracy. It is noted that several applications require mining the transaction data to capture the customer behavior frequently. In those applications, the efficiency of data mining could be a more important factor than the requirement for complete accuracy of the mining results. Allowing imprecise results can significantly improve the data mining efficiency. In this paper, two methods for mining association rules with adjustable accuracy are developed. By dealing with the concept of sampling, both methods obtain some essential knowledge from a sampled subset first, and in light of that knowledge, perform efficient association rule mining on the entire database. A technique of relaxing the support factor based on the sampling size is devised to achieve the desired level of accuracy. These two methods differ from each other in their ways of utilizing the sampled data. Performance of these two methods is comparatively analyzed. As shown by our experimental results, the relaxation factor, as well as the sample size, can be properly adjusted so as to improve the result accuracy while minimizing the corresponding execution time, thereby allowing us to effectively achieve a design trade-off between accuracy and efficiency with two control parameters. It is shown that with the advantage of controlled sampling, the proposed methods are very flexible and efficient, and can in general lead to results of a very high degree of accuracy.

1 Introduction

Due to the increasing use of computing for various applications, the importance of data mining is growing at a rapid pace recently. Progress in bar-code technology has made it possible for retail organizations to collect and store massive amounts of sales data. Catalog companies can also collect sales data from the orders they received. It is noted that analysis of past transaction data can provide very valuable information on customer buying behavior, and thus improve the quality of business decisions. In essence, it is necessary to collect and analyze a sufficient amount of sales data before any meaningful conclusion can be drawn therefrom. Since the amount of these processed data tends to be huge, it is important to devise efficient algorithms to conduct mining on these data.

Various data mining capabilities have been explored in the literature. Among them, the one receiving the most amount of research attention is on mining association rules [2, 3, 5, 8, 9, 12, 13, 14]. For example, given a database of sales transactions, it is desirable to discover all associations among items such that the presence of some items in a transaction will imply the presence of other items in the same transaction. Also, mining classification is an approach of trying to develop rules to group data tuples together based on certain common features. This has been explored both in the AI domain [10, 11] and in the context of databases [4, 6].

In general, data mining is a very application-dependent issue and different applications explored will require different mining techniques to cope with. Notice, however, that in several data mining applications, the problem domain could only be vaguely defined, and hence a mathematical formulation of the problem is usually called for to abstract the original problem. As such, the corresponding parameter selection is often subject to interpretation. For example, to address the problem of identifying consumer buying patterns, a mathematical model of mining association rules is formulated. Based on this formulation, purchasing a group of m items, $\{A_1, ..., A_m\}$, is said to imply a high likelihood of concurrently purchasing another item B, if (1) the group, $\{A_1, ..., A_m\}$, appears in at least $s\%$ of the transactions, and (2) among the transactions including $\{A_1, ..., A_m\}$, $c\%$ of them also include item B.[1] In the literature [3, 9], s and c are referred to as the minimum *support* and *confidence* levels of the association rule, respectively. The selection of parameter values for s and c usually has to be based on experience or even resorts to try-and-error. Moreover, another challenging issue is on how items should be grouped together in order to make the counting of occurrences meaningful, as studied in [7, 13].

In this paper, we devise efficient algorithms for mining association rules with adjustable accuracy to the mathematical formulation. It is noted that several applications require mining the transaction data to capture the customer behavior frequently. In those applications, the efficiency of data

* J. S. Park is partially supported by KOSEF, Korea.

CIKM 97 Las Vegas Nevada USA
Copyright 1997 ACM 0-89791-970-x 97/11..$3.50

[1] More details on the model of mining association rules are given in Section 2.

mining could be a more important factor than the requirement for complete accuracy of the results according to the mathematical formulation. Consider again the previous example of mining consumer purchase pattern. Since the way that confidence (c) and support (s) factors are selected is not very precise, i.e., somewhat experimental, missing some marginal cases with confidence and support levels at the borderline may have little effect on the quality of the solution to the original problem. Allowing imprecise results can in fact significantly improve the efficiency of the mining algorithms. In this paper, two methods for mining association rules with adjustable accuracy are developed. By dealing with the concept of sampling, both methods obtain some essential knowledge from a sampled subset first, and in light of that knowledge, perform efficient association rule mining on the entire database. Specifically, a technique of relaxing the support factor based on the sampling size is devised to achieve the desired level of accuracy. While both employing this technique, these two methods differ from each other in their ways of utilizing the sampled data. Performance of these two methods is comparatively analyzed. As will be seen later, the relaxation factor, as well as the sample size, can be properly adjusted so as to improve the result accuracy while minimizing the corresponding execution time. Thus, one can effectively achieve a design trade-off between accuracy and efficiency with two control parameters. Sensitivity analysis on several parameters is conducted. From our results, it is shown that with the advantage of controlled sampling, the proposed methods are very flexible and efficient, and can in general lead to results of a very high degree of accuracy.

We comment that sampling has been used in [13] for determining the cut-off level in the class hierarchy of items to collect occurrence counts in mining generalized association rules. The algorithm in [13] is able to detect any subsequent sampling error and make corrections. However, sampling is used there as a means to improve efficiency, rather than a means to achieve a trade-off between accuracy and efficiency. Sampling has also been discussed in [8] as a justification for devising algorithms and conducting experiments with data sets of small sizes. Concurrent to our work, a sampling method to find large itemsets has been proposed in [14] for reducing the number of passes through the database using a lowered frequency threshold, i.e., relaxing support. The algorithm in [14] computes all large itemsets in a sample and then confirms all the itemsets by scanning the rest of the database, while our approach uses a sample to find just large 1-itemsets, which will be explained in detail. By incorporating the technique of relaxing support into the sampling method, the emphasis of this paper is on optimizing the trade-off between execution efficiency and result accuracy. As shown by our experimental results, dealing with both the sample size and the relaxation factor, the proposed methods are capable of achieving a much better design trade-off between efficiency and accuracy than the prior work that relied upon adjusting the sample size only. As the database size increases and sampling appears to be an attractive approach for data mining nowadays, the methods devised in this paper improve not only the flexibility but also the practicality of the sampling approach.

The rest of the paper is organized as follows. Preliminaries are given in Section 2. The two sampling schemes are described in Section 3. Performance studies on various schemes are conducted in Section 4 via simulation. In Section 5, remarks on the relationship between two control parameters, sample size and relaxation factor, are made. This paper concludes with Section 6.

Database D

TID	Items
100	A C D
200	B C E
300	A B C E
400	B E

Figure 1: An example transaction database for data mining

2 Preliminaries

Since the algorithms proposed in this paper are developed by incorporating the sampling technique into algorithm DHP presented in [9], and DHP is a revised version of Apriori reported in [3], we shall briefly describe Apriori and DHP in this section. Readers interested in more details of these two prior algorithms are referred to [3] and [9].

2.1 Mining Association Rules by Apriori

Let $\mathcal{I}=\{i_1, i_2, ..., i_m\}$ be a set of literals, called items. Let D be a set of transactions, where each transaction T is a set of items such that $T \subseteq \mathcal{I}$. Note that the quantities of items bought in a transaction are not considered, meaning that each item is a binary variable representing if an item was bought. Each transaction is associated with an identifier, called TID. Let X be a set of items. A transaction T is said to contain X if and only if $X \subseteq T$. An association rule is an implication of the form $X \Longrightarrow Y$, where $X \subset \mathcal{I}$, $Y \subset \mathcal{I}$ and $X \bigcap Y = \phi$. The rule $X \Longrightarrow Y$ holds in the transaction set D with *confidence* c if c% of transactions in D that contain X also contain Y. The rule $X \Longrightarrow Y$ has *support* s in the transaction set D if s% of transactions in D contain $X \bigcup Y$.

The problem of mining association rules is composed of the following two steps:

1. Discover the large itemsets, i.e., all sets of itemsets that have transaction support above a pre-determined minimum support s.
2. Use the large itemsets to generate the association rules for the database.

The overall performance of mining association rules is in fact determined by the first step. After the large itemsets are identified, the corresponding association rules can be derived in a straightforward manner. Apriori, DHP, and also the sampling methods proposed in this paper mainly address the issue of discovering large itemsets.

Using an example transaction database given in Figure 1[2], we now describe the method used in Apriori for discovering large itemsets. In Apriori [3], in each iteration (or each pass) it constructs a candidate set of large itemsets, counts the number of occurrences of each candidate

[2]This example database is extracted from [3].

itemset, and then determines large itemsets based on a predetermined minimum support. In the first iteration, Apriori simply scans all the transactions to count the number of occurrences for each item. The set of candidate 1-itemsets, C_1, obtained is $\{\{A\},\{B\},\{C\},\{D\},\{E\}\}$. Assuming that the minimum transaction support required is 2 (i.e., $s = 40\%$), the set of large 1-itemsets, L_1, composed of candidate 1-itemsets with the minimum support required, can then be determined. Then, $L_1 = \{\{A\},\{B\},\{C\},\{E\}\}$. To discover the set of large 2-itemsets, in view of the fact that any subset of a large itemset must also have minimum support, Apriori uses $L_1 * L_1$ to generate a candidate set of itemsets C_2 where $*$ is an operation for concatenation in this case.[3] $C_2 = \{\{AB\}, \{AC\}, \{AE\}, \{BC\}, \{BE\}, \{CE\}\}$. Next, the four transactions in D are scanned and the support of each candidate itemset in C_2 is counted. The set of large 2-itemsets, L_2, is therefore determined based on the support of each candidate 2-itemset in C_2. Then, $L_2 = \{\{AC\}, \{BC\}, \{BE\}, \{CE\}\}$.

The set of candidate itemsets, C_3, is generated from L_2 as follows. From L_2, two large 2-itemsets with the same first item, such as $\{BC\}$ and $\{BE\}$, are identified first. Then, Apriori tests whether the 2-itemset $\{CE\}$, which consists of their second items, constitutes a large 2-itemset or not. Since $\{CE\}$ is a large itemset by itself, we know that all the subsets of $\{BCE\}$ are large and then $\{BCE\}$ becomes a candidate 3-itemset. There is no other candidate 3-itemset from L_2. Apriori then scans all the transactions and discovers the large 3-itemsets L_3. Since there is no candidate 4-itemset to be constituted from L_3, Apriori ends the process of discovering large itemsets.

2.2 Algorithm DHP

Same as Apriori, DHP also generates candidate k-itemsets from L_{k-1}. However, DHP is unique in that it employs a hash table, which is built in the previous pass, to test the eligibility of a k-itemset. Instead of including all k-itemsets from $L_{k-1} * L_{k-1}$ into C_k, DHP adds a k-itemset into C_k only if that k-itemset is hashed into a hash entry whose value is larger than or equal to the minimum transaction support required. DHP also reduces the database size progressively by not only trimming each individual transaction size but also pruning the number of transactions in the database. We note that both DHP and Apriori are iterative algorithms on the large itemset size in the sense that the large k-itemsets are derived from the large $(k-1)$-itemsets.

An example of generating candidate itemsets by DHP is given in Figure 2. For the candidate set of large 1-itemsets, which is $C_1 = \{\{A\},\{B\},\{C\},\{D\},\{E\}\}$ in the example, all transactions of the database are scanned to count the support of these 1-itemsets. For each transaction, after occurrences of all the 1-itemsets are counted, all the 2-subsets of this transaction are generated and hashed into a hash table H_2 in such a way that when a 2-itemset is hashed to bucket i, the value of bucket i is increased by one. Figure 2 shows a hash table H_2 for a given database. After the database is scanned, each bucket of the hash table has the number of 2-itemsets hashed to the bucket. Given the hash table in Figure 2 and a minimum support equal to 2, we can filter out candidate 2-itemsets from $L_1 * L_1$, to obtain $C_2 = \{\{AC\},\{BC\},\{BE\},\{CE\}\}$, instead of $C_2 =$

[3] For the general case, $L_k * L_k = \{X \cup Y | X, Y \in L_k, |X \cap Y| = k-1\}$.

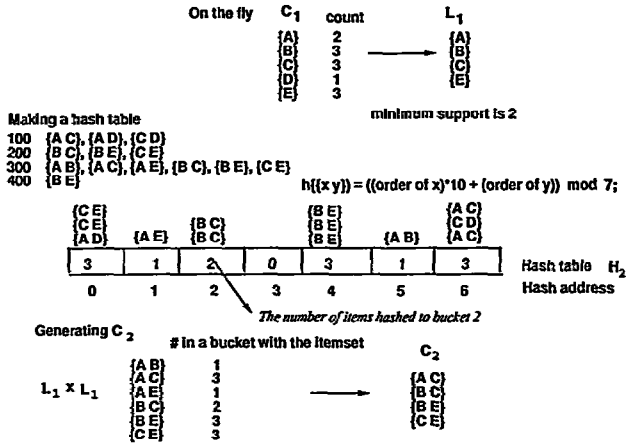


Figure 2: Example of a hash table and generation of C_2

$\{\{AB\}, \{AC\}, \{AE\}, \{BC\}, \{BE\}, \{CE\}\}$ resulted by Apriori. As shown in [9], the hash filtering in DHP can drastically reduce the size of C_k for small k values, especially for $k = 2$, and lead to prominent performance improvement.

3 Mining Association Rules with Adjustable Accuracy

In this section, the two proposed methods for mining association rules with adjustable accuracy are described. These two methods explore the trade-off between execution efficiency and result accuracy. A technique of relaxing the support factor based on the sampling size is utilized by both methods to effectively achieve the desired level of accuracy.

Due to the nature of sampling, results may not be completely accurate. While employing the same notation as in prior work described in Section 2, we put a "prime" to the symbol of a data set to represent a data set from the sampling methods that may be inaccurate. For example, while L_k is the *real* set of large k-itemsets, L'_k means the set of large k-itemsets from the sampling methods. The meanings of H'_k and C'_k are defined similarly.

3.1 Method on Based on Direct Sampling

The first method, performing association rule mining by utilizing direct sampling, is referred to as algorithm DS. First, according to a subset of the transaction database, DS determines L'_1 and H'_2 from the subset. The minimum support required for a 1-itemset to be included into L'_1 is set to αs where s is the given minimum support for large itemsets, and α, whose value is within [0,1], is the relaxation factor. Since αs is smaller than s, this L'_1 tends to be larger than the real L_1. As will be shown in Section 4 later, the relaxation factor, as well as the sample size, can be properly adjusted so as to improve the result accuracy and to minimize the corresponding execution time.

After L'_1 and H'_2 are obtained, DS then generates the candidate 2-itemsets, C'_2, from L'_1 and H'_2. From the description in Section 2, it is noted that large k-itemsets are

derived from large $(k-1)$-itemsets. That is, for an item to appear in a large k-itemset, this item must have appeared in a large $(k-1)$-itemset. In other words, if an item is not in L_1 then this item can be removed from consideration for all later construction for L_k, $k > 1$. This concept is incorporated into both of our schemes and found to be able to achieve much better execution efficiency[4]. Explicitly, when generating the candidate 2-itemsets C_2' from L_1' and H_2', we identify from L_1' those items which are not used to generate C_2'. Such a set of identified items is denoted by $\bar{\mathcal{F}}$, and is employed as a filter to improve the efficiency of large itemset generation[5]. The use of a set of large 1-itemsets as a filter to improve the generation of candidate large 2-itemsets is the very reason that DS is designed to obtain L_1' from the sample database first.

Next, after C_2' is obtained DS scans the whole transaction database to obtain the real L_1, and also to determine L_2' from C_2'. If an item of a transaction belongs to $\bar{\mathcal{F}}$, then this item is filtered out for the formation of any 2-itemset in L_2'. Finally, the remaining steps to find L_k, for $k \geq 3$, are same as those in DHP. It can be seen that since the whole database is scanned by DS to determine L_k', for $k \geq 2$, the resulting L_k' is a subset of L_k for $k \geq 2$.

The procedure of DS is summarized below, and its flowchart is given in Figure 1 (a) for illustrative purposes.

Algorithm DS: Mining association rules based on direct sampling.

Step 1: Determine L_1' and H_2' from a subset of the transaction database with a minimum support αs.

Step 2: Generate the candidate 2-itemsets, C_2', from L_1' and H_2'. Let $\bar{\mathcal{F}}$ be a set of items which are not used to generate C_2'.

Step 3: Scan the whole transaction database to decide the real L_1 and also obtain L_2' from C_2'. If an item of a transaction belongs to $\bar{\mathcal{F}}$, then this item is filtered out for the formation of any 2-itemset in L_2'.

Step 4: Perform the remaining steps the same as the steps of DHP to find L_k for $k \geq 3$.

3.2 Method on Based on Sampling with Effective Hash Construction

The second method, refereed to as algorithm SH, performs association rule mining by sampling with effective hash table construction. Algorithm SH utilizes the same underlying concept as DS, but is different from the latter in their ways of utilizing the sampled data. Algorithm SH is outlined as follows. First, according to a subset of the transaction database, SH determines L_1' with a minimum support αs, and obtains a set of items $\bar{\mathcal{F}}$ which is composed of items not belonging to L_1'. Then, SH scans the whole transaction database to decide the real L_1 and H_2'. While all the items in each transaction are used to decide L_1, items belonging to $\bar{\mathcal{F}}$ are filtered out for the generation of H_2'. C_2' is then generated from L_1 and H_2', and those items which are not used to generate C_2' are added to $\bar{\mathcal{F}}$. Note that as the number of items in $\bar{\mathcal{F}}$ increases, its filtering effect improves. Next, using $\bar{\mathcal{F}}$ as a filter, SH scans the whole transaction database to decide L_2'. Finally, the remaining steps to find L_k, for $k \geq 3$, are same as those in DHP. Same as in DS, it can be seen that the resulting L_k' from SH is a subset of L_k, for $k \geq 2$. The procedure of SH is summarized below with the corresponding flowchart given in Figure 1 (b).

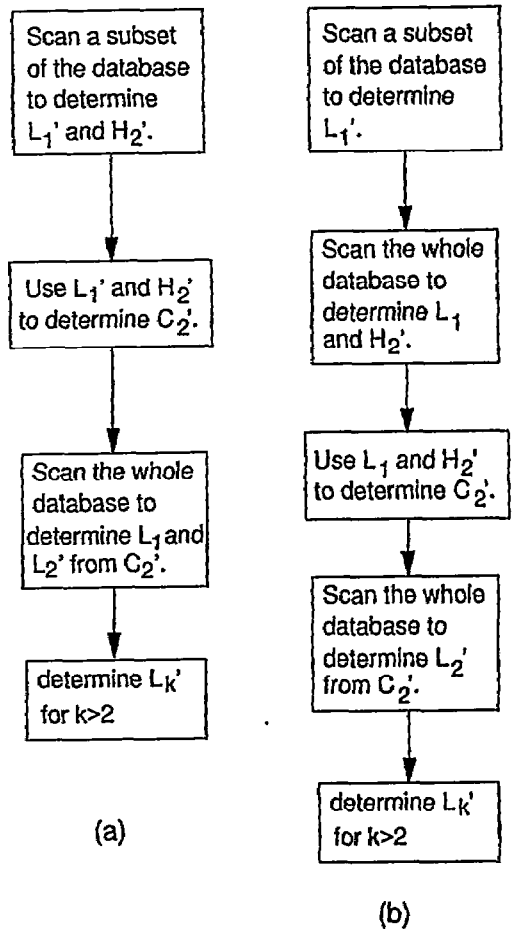


Figure 3: Flowcharts of sampling methods: (a) DS and (b) SH.

Algorithm SH: Mining association rules based on sampling with effective hash table construction.

Step 1: Determine L_1' from a subset of the transaction database with a minimum support αs. Obtain a set of items $\bar{\mathcal{F}}$ which is composed of items not belonging to L_1'.

Step 2: Scan the whole transaction database to decide the real L_1 and H_2'. Filter out items belonging to $\bar{\mathcal{F}}$ for the generation of H_2'.

Step 3: Generate C_2' from L_1 and H_2'. Those items which are not used to generate C_2' are added to $\bar{\mathcal{F}}$.

Step 4: By using $\bar{\mathcal{F}}$ as a filter, scan the whole transaction

[4]Note that this filtering technique is also applicable to algorithm Apriori.

[5]As can be seen later, we implemented this concept by inverse filtering, i.e., items falling into $\bar{\mathcal{F}}$ will be removed from later consideration. Clearly, one could implement it as normal filtering, i.e., only items in $\mathcal{F}$ will require further processing.

database to decide L'_2.

Step 5: Perform the remaining steps the same as the steps of DHP to find L_k for $k \geq 3$.

From Figure 1, it can be observed that the very difference between DS and SH stems from the feature that SH obtains its L'_1 from its sample set first without generating H'_2 at the same time. Explicitly, while DS utilizes the L'_1 obtained from the sample data to improve the efficiency of generating C'_2, SH utilizes the L'_1 from the sample data to improve the efficiency of generating H'_2. (The H'_2 of DS is obtained together with L'_1 from the sample data.) It can also be seen that, as a consequence, SH incurs one more run of database scan than DS. The reason that SH is so designed is explained below. It is found from DS that with the performance improvement achieved, the generation of H'_2 could be costly itself. To improve the efficiency of generating H'_2, SH gets a set of large 1-itemsets from a sample database first (in Step 1) and uses it as a filter to facilitate the generation of H'_2. In contrast to the H'_2 of DS which is obtained together with L'_1 from a sample data, the H'_2 of SH is obtained with a full database scan (in Step 2) and is thus more accurate and effective than that of DS. Clearly, the above advantage of SH has to be compensated with its extra run of database scan. Performance of DS and SH is comparatively analyzed in Section 4. In general, it is shown that DS incurs shorter execution time while SH leads to more accurate results.

3.3 Quality of Results from the Proposed Methods

While achieving better execution efficiency, the large itemsets obtained by the proposed methods may not be complete accurate. Though DS and SH are designed for applications allowing inaccurate results, it is still important to understand the degree of inaccuracy resulted. In our discussion, two types of inaccuracy are examined. First, if a set of large itemsets by the sampling method, i.e., L'_k, is not identical to the real set of large itemsets, i.e., L_k (as obtained by DHP), then we say there is a *discrepancy* between these two sets of large itemsets. On the other hand, if a complete set of results (containing all L'_k, for $k \geq 1$) from the sampling method is not exactly the same as the set of all L_k, for $k \geq 1$, i.e., if there exists an m such that there is a discrepancy between L'_m and L_m, then we say there is a *mismatch* between these two sets of results. The corresponding experiment is then called a case with mismatch. To assess the performance of proposed methods, extensive experimental studies will be conducted in Section 4 to analyze both mismatch and discrepancy resulting from DS and SH.

For illustrative purposes, a case with mismatch is presented below, which is an example run of algorithm DS[6]. In this experiment, there are 100,000 transactions in the database and the minimum support required is 5% (i.e., 5,000 transactions). Also, the sample size used is 10% of the entire transaction database (i.e., 10,000 transactions) and the relaxation factor α used is 0.75. By running DHP and DS, we get $|L'_2| = 1892 < |L_2| = 1897$,
$|L'_3| = 2086 < |L_3| = 2089$,
$|L'_4| = 1846 < |L_4| = 1847$,

[6]The results from an example run of SH contain similar information and are thus omitted here. An empirical study on both DS and SH is conducted in Section 4.

Table 1: Missed 2-itemsets and their counts

Rank (in 1897)	2-itemset missed	count
1548^{th}	(257, 616)	571
1671^{th}	(356, 713)	543
1693^{th}	(257, 664)	539
1807^{th}	(693, 934)	517
1870^{th}	(257, 698)	503

$|L'_5| = |L_5| = 1300$, $|L'_6| = |L_6| = 725$, $|L'_7| = |L_7| = 310$, $|L'_8| = |L_8| = 97$, $|L'_9| = |L_9| = 20$, and $|L'_{10}| = |L_{10}| = 2$. It can be seen that in this experiment with mismatch, there are discrepancies between L'_i and L_i for $2 \leq i \leq 4$, whereas L'_i and L_i are the same for $5 \leq i \leq 10$. Recall that in the process of determining large itemsets, each itemset receives a support count for its occurrences, and is declared to be a large itemset if that count meets the minimum support required. Note that given a pair of (L'_j, L_j) with discrepancy, in addition to knowing that there are some large j-itemsets are missing in L'_j, it is important to learn the relative importance of those itemsets missing, i.e., what are the support counts for those missed itemsets. Obviously, missing an itemset with a large support count is more undesirable than missing one with a small count.

In this experiment, itemsets in L_i, $2 \leq i \leq 4$, are sorted based on their supports in descending order, and those which are absent in L'_i are identified. Missed large 2-itemsets and their support counts are given in Table 1. From Table 1, it can be seen that the amount of discrepancy between L'_2 and L_2 is fact less than 0.2% (i.e., $\frac{4}{1897}$), and also that the 5 large itemsets missing in L'_2 are mostly in the bottom 10% of those in L_2 in view of their support counts, meaning that the itemsets missed by DS are in the borderline and of relatively less importance than other large itemsets. This agrees with our intuition since an itemset with a strong support count is very unlikely to be overlook by L'_1 of DS with a relaxed support level αs. Missed large 3-itemsets and large 4-itemsets are shown in Tables 2 and 3, respectively. Again, the amount of discrepancy between L'_3 and L_3 and that between L'_4 and L_4 are both less than 0.2%, and the itemsets missing are all in the bottom level in accordance with their support counts.

It is noted that while generating the real L_1, DS obtains L'_k that is a subset of L_k, for $2 \leq k$, meaning that no false association rules would be output by DS. Moreover, since large itemsets with strong support counts are captured by DS, the association rules missed by DS, if any, are in general borderline cases and are of less importance. The same qualitative assessment also holds for SH. It can be observed from Tables 1, 2 and 3 that the overlook of the item 257 by L'_1 in Step 1 of DS indeed causes a few subsequent large itemsets missing (three in L'_2, three in L'_3 and one in L'_4). As will be shown later, the accuracy of capturing large 1-itemsets into L'_1 by both sampling methods DS and SH can be properly adjusted by dealing with the sample size and the relaxation factor employed.

Table 2: Missed 3-itemsets and their counts

Rank (in 2089)	3-itemset missed	count
1634^{th}	(257, 566, 616)	564
2000^{th}	(257, 566, 664)	513
2017^{th}	(257, 616, 664)	511

Table 3: Missed 4-itemsets and their counts

Rank (in 1847)	4-itemset missed	count
1793^{th}	(257, 566, 616, 664)	510

4 Experimental Results

To assess the performance of proposed methods, we conducted several experiments on finding large itemsets using a SUN SPARC/10 workstation with a CPU clock rate of 40MHz running Solaris 2.3. Section 4.1 explains the way that the synthetic transaction workload is generated, and Section 4.2 compares the performance of four different mining methods including DS, SH, DHP and Apriori. The percentage of cases with mismatches and the fractional amount of discrepancy caused by the sampling approach are analyzed in Sections 4.3 and 4.4, respectively. Finally, the effect of relaxation factor is studied in Section 4.5.

4.1 Generation of Synthetic Workload

Synthetic transactions are generated from an N item set with the objective of emulating sales transactions in a retail industry. The method of creating the transaction database is in essence similar to those used in [3] and [9]. We generate 40 different transaction databases from the same set of potentially large itemsets to evaluate the performance of the two sampling methods, DS and SH. Each database consists of $|D|$ transactions, and on the average each transaction has $|T|$ items. Table 4 summarizes the meaning of various parameters used in the experiments. Figure 4 shows the process used in generating the transaction database, which is briefly described below. More details on the workload

Table 4: Meaning of various parameters.

D_k	Set of transactions for large k-itemsets.
$\|D_k\|$	The number of transactions in D_k.
H_k	Hash table containing $\|H_k\|$ buckets for C_k.
C_k	Set of candidate k-itemsets.
L_k	Set of large k-itemsets.
$\|D\|$	Number of transactions in the database.
$\|T\|$	Average size of the transactions.
$\|I\|$	Average size of the maximal potentially large itemsets.
$\|L\|$	Number of maximal potentially large itemsets.
N	Number of items.

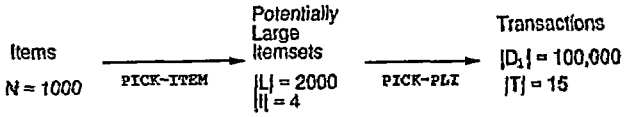


Figure 4: The process of generating a single synthetic transaction file

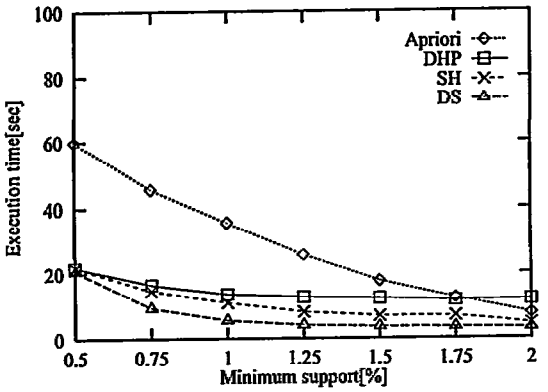
generation can be found in [3] and [9]. Synthetic transactions consist of a series of potentially large itemsets (PLI's) which are chosen according to the weight of each PLI. The method of selecting PLI's to form a transaction is referred to as PICK-PLI in Figure 4. The weight of each PLI corresponds to the probability that this PLI will be included into the synthetic transactions. A weight of each PLI is picked from an exponential distribution with unit mean. Furthermore, all the weights are normalized so that the sum of them is equal to one. For the first PLI in PICK-ITEM of Figure 4, items are chosen by a random selection method where a uniform distribution is used to select an item out of N items. Some fraction of the items in subsequent PLI's is chosen from the previous PLI and the remaining items are chosen based on the random selection method. The common fraction is referred to as the correlation level and is generated from an exponentially distributed random variable with the mean equal to a predetermined value. The mean of the correlation level is set to 0.25 for our experiments.

4.2 Comparison of Four methods

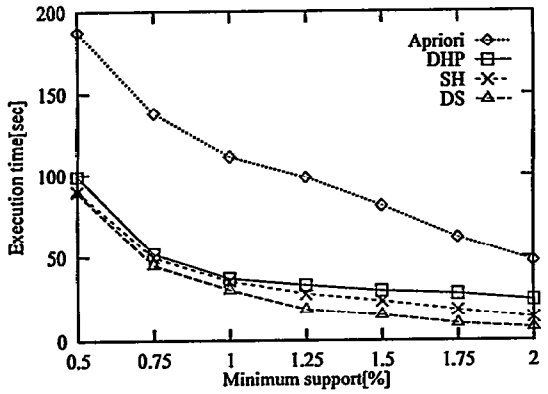
Figure 5 shows the experimental results of four different mining methods for finding large itemsets. The results in Figure 5 are associated with the case when $N = 1000$, $|L| = 2000$, $|I| = 4$, and $|D| = 100,000$. The sample size for the sampling methods, DS and SH, is 10% of the transactions. In addition, until analyzed in Section 4.5, the relaxation factor α is set to be 0.75 in our experiments. Apriori in Figure 5 refers to algorithm Apriori in [3], and DHP refers to algorithm DHP in [9]. In Figure 5, it can be observed that the two sampling methods DS and SH achieve shorter execution time than DHP and Apriori for various minimum support levels under two different transaction sizes, i.e., $|T|$=10 and $|T|$=15. The results show that the execution time of DS is smaller than one-third of that of DHP for 2.0% minimum support, showing significant performance improvement. Among the two sampling methods, DS outperforms SH for various minimum support levels.

Comparing DHP with DS, we note that in order to generate hash table H_2, algorithm DHP considers all combinations of 2-items for each transaction in the database, while DS uses a subset of the transactions for generating H'_2. The time for generating H_2 in algorithm DHP is indeed a major part of its execution time. By improving the efficiency of generating H'_2, the total execution time of DS is reduced. In addition to reducing the time for generating H'_2, filtering items out by $\bar{\mathcal{F}}$ for generating L'_2 can also improve performance of the sampling method, because this also reduces the number of combinations of 2-items to be considered. For example, for the case of 0.5% minimum support in Figure 5 (b), $\bar{\mathcal{F}}$ contains 215 items among a set of 1000 items, and for 2.0% minimum support $|\bar{\mathcal{F}}|$ is approximately 880.

The sampling methods DS and SH are more efficient than Apriori and DHP whereas DS and SH may lose some ac-



(a) for $|T| = 10$



(b) for $|T| = 15$

Figure 5: Execution time to minimum supports.

curacy for low minimum supports. As we shall see later, proper selection of relaxation factor and sample size by DS and SH can lead to improvement in execution time even for the stringent case requiring (near) complete accuracy. In fact, both DS and SH in Figure 5 (a) generate all real large itemsets (i.e. complete accuracy) when minimum supports are larger than 0.75%. In the remaining figures of Section 4, results on the accuracy of these methods will be shown for $|T| = 15$.

We now take a closer look at the difference between DS and SH. Although DS achieves smaller execution time than SH, it can also incur a higher level of inaccuracy for cases with low minimum supports. The ratio between cardinalities of these two sets in each pass will be studied in Section 4.4. Finally, we observe that the execution time of DHP for 2.0% minimum support in Figure 5 (a) is worse than that of Apriori. This is due to the fact that all large itemsets satisfying the minimum support in this case are 1-itemsets. At the first pass, Apriori generates simply large 1-itemsets, whereas DHP generates large 1-itemsets and the hash table H_2. For cases where results involve large k-itemsets for $k \geq 2$, DHP in general outperforms Apriori.

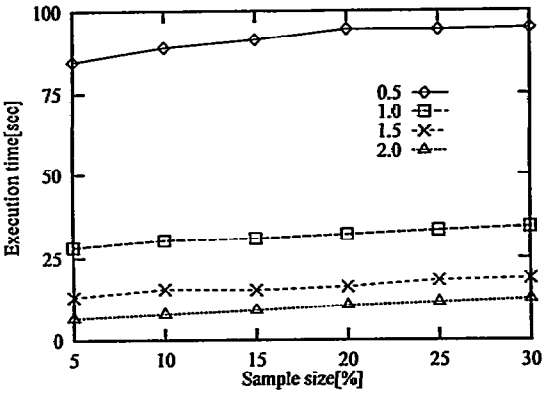


Figure 6: Execution time to sample sizes for DS.

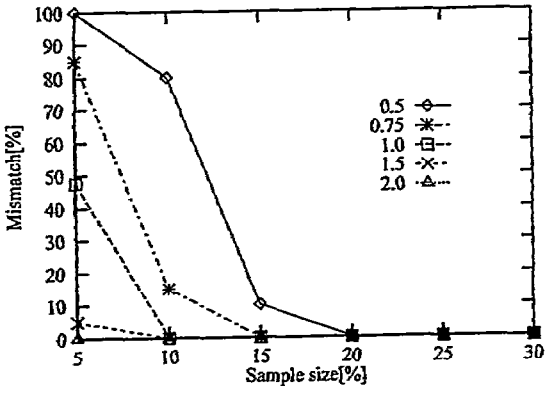
4.3 The Percentage of Cases with Mismatches

Figure 6 shows the execution times of DS by varying sample sizes under four different minimum support levels, ranging from 0.5% to 2.0%. The case with a 0.5% minimum support is labeled with 0.5 in the figure. The other support levels are labeled similarly. As shown in Figure 6, the execution time increases as the sample size grows. This is due to the reason that with a larger sample size more transactions have to be processed at the first step to generate L'_1 and H'_2. (A similar phenomenon is also observed for SH.) As the sample size reduces, the accuracy decreases as well. The trade-off between the sample size and the result accuracy is studied below.

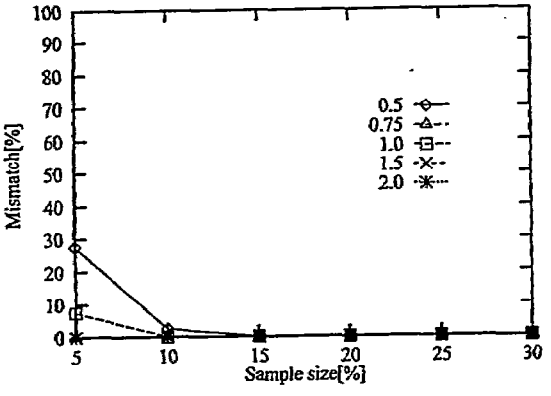
As mentioned before, we generate 40 transaction databases to compare the two sampling methods DS and SH. The sampling method may miss certain number of *real* large itemsets for some of these 40 cases. The percentage of cases with mismatch is used as a means to measure the accuracy of the sampling method. As will be addressed in Section 4.4, the other measure considered is the amount of discrepancy between the set of large itemsets generated by the sampling method and the set of real large itemsets.

Let the percentage of cases with mismatches, ϵ_m, be the ratio of the number of cases with mismatched results from a sampling method to the total number of cases investigated. Figure 7 shows the percentages of mismatches by varying sample sizes under 5 different minimum supports for DS and SH. As expected, the percentage of cases with mismatches decreases as the sample size increases. In Figure 7, for most cases the value of ϵ_m drops rapidly as the sample size changes from 5% to 10%. In general, SH obtains more accurate results than DS, whereas the latter incurs shorter execution time (as shown in Figure 5). Since DS generates L'_1, H'_2, and C'_2 from a sample of transactions, the percentages of mismatches of DS are relatively larger than those of SH which uses L'_1 to filter out some items from H'_2.

Note that though the sampling methods can introduce inaccuracy for low minimum supports and sample sizes, it is very useful to obtain large itemsets efficiently, in particular when one wants to know an approximate set of large itemsets. When the minimum support is larger than or equal to 1.0%, the value of ϵ_m for both DS and SH is zero even for



(a) Algorithm DS



(b) Algorithm SH

Figure 7: The percentage of cases with mismatches ϵ_m to sample sizes.

the case that the sample size is only 10% of the transaction database.

4.4 The Fractional Amount of Discrepancy

In the previous subsection, we observe that DS can have more cases with mismatches. We now take a closer look at how serious is the discrepancy, i.e. how many large itemsets are missed due to the fast sampling method. The amount of discrepancy shown below is averaged over the cases with mismatches. In other words, it is the expected amount of discrepancy given that we know a mismatch occurs. As pointed out earlier, neither DS nor SH would falsely introduce large itemsets. They could only miss some large itemsets. Since a L'_k is always a subset of L_k, the fractional amount of discrepancy ϵ_d is defined as $\frac{|L_k|-|L'_k|}{|L_k|}$.

Figure 8 shows the fractional amount of discrepancy between L'_k by DS and the real L_k for each pass k. The figure also shows how much of the fractional amount of discrepancy in L'_k propagates to the next pass to affect the generation

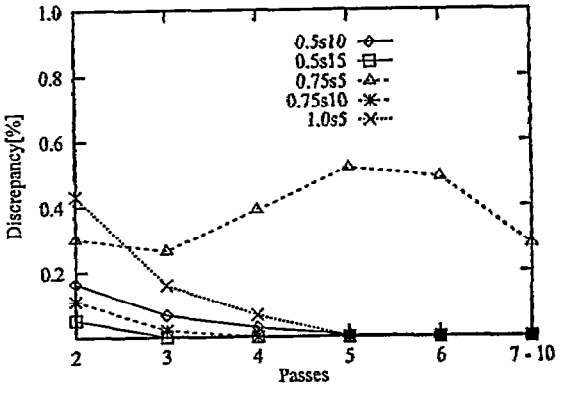


Figure 8: The fractional amount of discrepancy to each pass for DS.

of L'_{k+1} for $k \geq 2$. In Figure 8, we label each curve as xsy where x is the minimum support in percentage, and y is the sample size in percentage of the transaction database. We tested five sets of minimum supports and sample sizes for the fractional amount of discrepancy. Recall that each value shown in the figure is the average value of fractional amounts of discrepancy over the cases which produced mismatched results by DS. Hence, if such an average is taken over the total number of cases, the fractional amount of discrepancy will be much smaller than the values shown in Figure 8.

The fractional amount of discrepancy decreases as the pass number increases, except for the case with a small sample size (i.e., 5%). This means that the propagation of the fractional amount of discrepancy is very low for most sample sizes, confirming with our observation from an example run of DS in Section 3.3. Figure 9 shows that if the sample size is greater than 10%, the fractional amount of discrepancy drops nearly to zero as the pass number approaches to 5.

4.5 Effect of Relaxation factor

Figure 9 shows the effect of relaxation factor for 10% sample size and various minimum supports. In Figure 9, the ordinate is the value of the relaxation factor and the abscissa represents the percentage of cases with mismatches, i.e., ϵ_m. In the first step of DS, we used the relaxation factor to relax the tight bound for a given minimum support. If αs is used instead of s for a minimum support, more 2-itemsets can be included in the candidate 2-itemsets C'_2, and this will make L'_2 almost equal to L_2, the real large 2-itemsets. When α is low, the candidate 2-itemsets C'_2 may contain the real large 2-itemsets L_2 and ϵ_m approaches zero. However, the corresponding execution time increases because of the larger size of C'_2. For example, when α changes from 0.6 to 0.75 for 0.75% minimum support, the number of candidate 2-itemsets reduces from 3208 to 2354, and the execution time changes from 49.29 sec to 46.36 sec, but ϵ_m deteriorates as its value increases from 0% to 15%.

Figure 10 shows the sensitivity of the fractional amount of discrepancy to relaxation factor. The percentage is computed from the cases with mismatches using the following

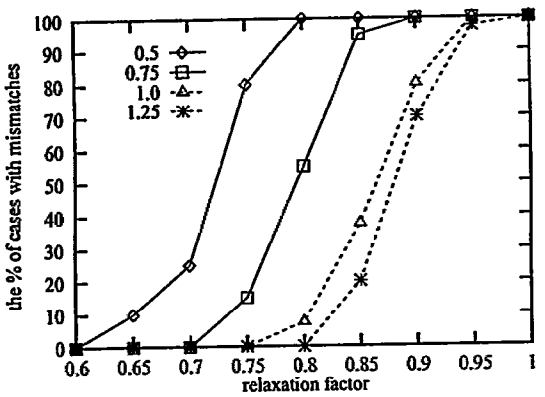


Figure 9: The percentage of cases with mismatches (ϵ_m) to relaxation factor (α)

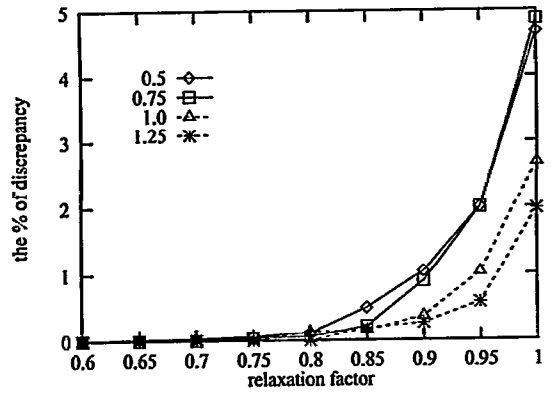


Figure 10: The fractional amount of discrepancy to relaxation factor (α)

equation:

$$\epsilon = \frac{\sum_{k=2}^{p} m_k}{\sum_{k=2}^{p} |L_k|},$$

where p is the maximal j such that the set of large j-itemsets is nonempty and m_k is the number of k-itemsets missed by DS. The percentage grows as the relaxation factor increases.

Figure 11 shows the execution time for varying relaxation factors under different minimum supports. As shown in Figures 10, 11 and 12, reducing the relaxation factor increases the execution time, albeit it improves the accuracy of the results.

5 Remarks on Relaxation Factor and Sample Size

We now examine the approach of *purely* sampling without relaxing the support level. We set $\alpha = 1$ and the minimum support to be 1% in our experiment here. As shown in Table

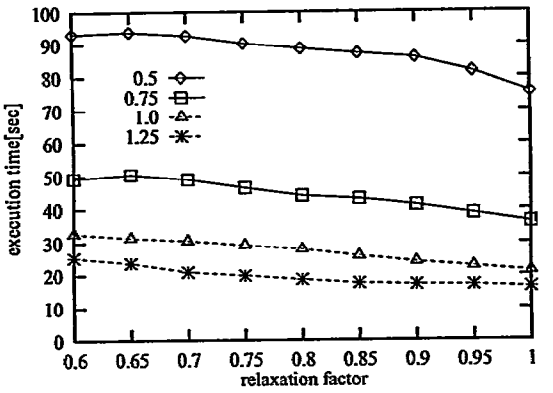


Figure 11: The execution time to relaxation factor (α)

5, even increasing the sample size to 50% of the database, the percentage of cases with mismatches is still 100%. Although increasing the sample size reduces the percentage of discrepancies, its effect seems to become marginal for sample size larger than 40%.

Next we use the method of DS with two different sample sizes and vary the relaxation factor α from 0.6 to 1.0 in Table 6. (Again the minimum support is 1%.) Let's consider the 5% sample size in Table 6. Clearly, the relaxation factor provides another means of controlling the accuracy. By setting α to be less than or equal to 0.8, we start to have at least one-fourth of the cases without mismatches, showing clear improvement. The execution time for $\alpha = 0.8$ is in fact shorter than that of the purely sampling approach with 30% sample size in Table 5, where the latter even incurs mismatches in all cases. This fact suggests that the approach of only controlling the sample size is inadequate.

Certainly, one can do a combination of increasing the sample size and reducing the relaxation factor to improve the accuracy while minimizing the increase in execution time. *It is critical to have both of these control knobs, sample size and relaxation factor, available in the sampling method.* There is a trade-off between settings of the two control knobs. Let's focus on the 10% sample size. Compared with the 5% sample size, we can see that even the stringent case of 0% mismatch can be achieved by *either* setting $\alpha = 0.6$ and using a 5% sample size *or* setting $\alpha = 0.75$ and using a 10% sample size. The latter in fact incurs shorter execution time. It also improves over the DHP method (See Figure 5 (b) which is for 10% sample size and 0.75 relaxation factor). *The results suggest that the proposed sampling approach be applicable to improving execution efficiency even for the cases requiring near* 0% *mismatch.*

The proper choice of relaxation factor and sample size can be obtained experimentally. Since the same mining application will be applied repeatedly, this off-line investment should be worthwhile. We note that although Chernoff bound has been mentioned in [8, 13, 14], it is too loose a theoretical bound to guide the selection of relaxation factor and sample size. The sample size implied by the Chernoff bound for a given level of accuracy and relaxation factor can be several orders of magnitude larger than what we observed in the experiment. Furthermore, it does not provide any clue on the impact on the efficiency (or execution time) of the

Table 5: DS with $\alpha = 1$

Sample size (%)	Execution Time (sec)	% of cases with Mismatches	% of Discrepancies
5	19.55	100	4.72
10	20.15	100	2.66
15	21.24	100	1.82
20	23.10	100	1.25
25	24.36	100	0.80
30	27.31	100	0.56
35	28.16	100	0.38
40	29.57	100	0.45
45	31.43	100	0.47
50	32.66	100	0.40

Table 6: DS with two sample sizes, 5% and 10%

α	Execution Time (sec)		% of cases with Mismatches		% of Discrepancies	
	5%	10%	5%	10%	5%	10%
0.6	31.48	32.82	0	0	0	0
0.65	30.69	31.43	5	0	0.16	0
0.7	29.97	30.58	17.5	0	0.15	0
0.75	28.53	29.15	47.5	0	0.18	0
0.8	26.21	27.86	72.5	7.5	0.28	0.13
0.85	24.39	25.79	100	37.5	0.64	0.15
0.9	23.02	24.01	100	80	1.29	0.36
0.95	20.87	22.56	100	100	2.97	1.02
1.0	19.58	21.02	100	100	4.7	2.66

different combinations of relaxation factors and sample sizes that provide the same level of accuracy.

6 Conclusion

We have devised and analyzed in this paper two algorithms for mining association rules with adjustable accuracy. By dealing with the concept of sampling, both methods obtained some essential knowledge from a sampled subset first, and in light of that knowledge, performed efficient association rule mining on the entire database. A technique of relaxing the support factor based on the sampling size was devised to achieve the desired level of accuracy. Performance of these two methods was comparatively analyzed. It is noted that the sample size and the relaxation factor can be adjusted so as to ensure the required accuracy of the results and to maximize the efficiency of mining. Sensitivity analysis on various parameters was conducted. From our results, it is shown that with the advantage of controlled sampling, the proposed methods are very flexible and efficient, and can in general lead to results of a very high degree of accuracy.

References

[1] R. Agrawal, S. Ghosh, T. Imielinski, B. Iyer, and A. Swami. An Interval Classifier for Database Mining Applications. *Proceedings of the 18th International Conference on Very Large Data Bases*, pages 560–573, August 1992.

[2] R. Agrawal, T. Imielinski, and A. Swami. Mining Association Rules between Sets of Items in Large Databases. *Proceedings of ACM SIGMOD*, pages 207–216, May 1993.

[3] R. Agrawal and R. Srikant. Fast Algorithms for Mining Association Rules in Large Databases. *Proceedings of the 20th International Conference on Very Large Data Bases*, pages 478–499, September 1994.

[4] T.M. Anwar, H.W. Beck, and S.B. Navathe. Knowledge Mining by Imprecise Querying: A Classification-Based Approach. *Proceedings of the 8th International Conference on Data Engineering*, pages 622–630, February 1992.

[5] S. Brin, R. Motwani, J. D. Ullman, and S. Tsur. Dynamic Itemset Counting and Implication Rules for Market Basket Data. *Proceedings of ACM SIGMOD*, pages 255–264, May, 1997.

[6] J. Han, Y. Cai, , and N. Cercone. Knowledge Discovery in Databases: An Attribute-Oriented Approach. *Proceedings of the 18th International Conference on Very Large Data Bases*, pages 547–559, August 1992.

[7] J. Han and Y. Fu. Discovery of Multiple-Level Association Rules from Large Databases. *Proceedings of the 21th International Conference on Very Large Data Bases*, pages 420–431, September 1995.

[8] H. Mannila, H. Toivonen, and A. Inkeri Verkamo. Efficient Algorithms for Discovering Association Rules. *Proceedings of AAAI Workshop on Knowledge Discovery in Databases*, pages 181–192, July, 1994.

[9] J. S. Park, M.-S. Chen, and P. S. Yu. An Effective Hash Based Algorithm for Mining Association Rules. *Proceedings of ACM SIGMOD*, pages 175–186, May, 1995.

[10] G. Piatetsky-Shapiro. Discovery, Analysis and Presentation of Strong Rules. *Knowledge Discovery in Databases*, pages 229–248, 1991.

[11] J.R. Quinlan. Induction of Decision Trees. *Machine Learning*, 1:81–106, 1986.

[12] A. Savasere, E. Omiecinski, and S. Navathe. An Efficient Algorithm for Mining Association Rules in Large Databases. *Proceedings of the 21th International Conference on Very Large Data Bases*, pages 432–444, September 1995.

[13] R. Srikant and R. Agrawal. Mining Generalized Association Rules. *Proceedings of the 21th International Conference on Very Large Data Bases*, pages 407–419, September 1995.

[14] H. Toivonen. Sampling Large Databases for Association Rules. *Proceedings of the 22nd International Conference on Very Large Data Bases*, pages 134–145, September 1996.