A New Two-Phase Sampling Based Algorithm for Discovering Association Rules

Bin Chen
Exelixis, Inc.
170 Harbor Way
S. San Francisco, CA 94083
bchen@exelixis.com

Peter Haas
IBM Almaden Research Ctr
650 Harry Road
San Jose, CA 95120-6099
peterh@almaden.ibm.com

Peter Scheuermann
Dept of Elec and Comp Engr
Northwestern University
Evanston, IL 600208
peters@ece.northwestern.edu

ABSTRACT

This paper introduces FAST, a novel two-phase sampling-based algorithm for discovering association rules in large databases. In Phase I a large initial sample of transactions is collected and used to quickly and accurately estimate the support of each individual item in the database. In Phase II these estimated supports are used to either trim "outlier" transactions or select "representative" transactions from the initial sample, thereby forming a small final sample that more accurately reflects the statistical characteristics (i.e., itemset supports) of the entire database. The expensive operation of discovering association rules is then performed on the final sample. In an empirical study, FAST was able to achieve 90–95% accuracy using a final sample having a size of only 15–33% of that of a comparable random sample. This efficiency gain resulted in a speedup by roughly a factor of 10 over previous algorithms that require expensive processing of the entire database — even efficient algorithms that exploit sampling. Our new sampling technique can be used in conjunction with almost any standard association-rule algorithm, and can potentially render scalable other algorithms that mine "count" data.

1. INTRODUCTION

The volume of electronically accessible data in warehouses and on the Internet is growing faster than the speedup in processing times predicted by Moore's Law [21]. Scalability of mining algorithms is therefore a major concern. Classical mining algorithms that require one or more passes over the entire database can take hours or even days to execute, and in the future this problem will only become worse.

One approach to the scalability problem is to exploit the fact that approximate answers often suffice, and execute mining algorithms over a "synopsis" or "sketch," that is, over a lossy compressed representation of the data. This approach can provide approximate answers while reducing the processing time by orders of magnitude — in the context of a rapidly-changing competitive environment, such quick approximate results can be much more useful than "exact" results that are irrelevant by the time they are computed. Use of a synopsis *per se*, however, is not guaranteed to solve the problem. The computation of many synopses proposed in the literature requires one or more expensive passes over all of the data, so that the use of these synopses may still fail to adequately address the scalability problem unless the cost of producing the synopsis is amortized over many queries.

Using a sample of the data as the synopsis is a popular technique that can scale very well as the data grows. Besides having desirable scaling properties, sampling is also well suited to interactive exploration of massive data sets [11]. Recent work in the area of approximate aggregation processing [1, 8, 10] shows that the benefits of sampling are most fully realized when the sampling technique is tailored to the specific problem at hand. In this spirit we initiate an investigation of sampling methods that are designed to work with mining algorithms for "count" datasets, that is, datasets in which there is a base set of "items" and each data element is a vector of item counts — here "items" may correspond to physical items, responses on a survey, income levels, and so forth. As a first step, we study sampling-based algorithms for the most well-studied mining problem defined on count data: the discovery of association rules in large transaction databases.

Agrawal, et al. [3] proposed association rules as a means of describing interesting purchasing patterns. Association rules identify relationships among sets of items and can be used to evaluate business trends, identify purchasing patterns, and classify customer groups. Two measures, called *support* and *confidence*, are introduced in [3] in order to quantify the significance of an association rule. The mining of association rules from a set of transactions is the process of identifying all rules having support and confidence greater than specified minimum levels; such rules are said to have "minimum confidence and support." We focus on the problem of finding the "frequent" itemsets that have minimum support, because this operation is by far the most expensive phase of the mining process. We assume that the reader is familiar with the basic Apriori algorithm introduced in [3]. A variety of modifications have been proposed to reduce the computational burden—see, for example, [2, 5, 7, 14] and references therein—but with few exceptions all current algorithms require at least one expensive pass over the data.

In the context of "standard" association-rule mining, use of samples can make mining studies feasible that were for-

SIGKDD '02 Edmonton, Alberta, Canada

merly impractical due to the enormous time requirements. Indeed, a number of large companies routinely run mining algorithms on a sample of their data rather than on the entire warehouse. Sampling-based algorithms also facilitate interactive mining. I.e., when the goal is to obtain one or more "interesting" association rules as quickly as possible, a user might first mine a very small sample. If the results are unsatisfactory, the sample size is then increased, and the mining step is executed again. The sample size is iteratively increased until interesting rules are found. This iterative procedure approximates the functionality of online systems such as those described in [11]. As indicated in the latter paper, user studies have shown that such online systems provide much higher levels of user satisfaction than systems based on traditional batch-style processing.

The primary challenge in developing sampling-based algorithms stems from the fact that the support of an itemset in a sample almost always deviates from the support in the entire database. (The support of an itemset is the fraction of transactions that contain the itemset.) Such luck-of-the-draw fluctuations can result in *missing* itemsets that are frequent in the database but not in the sample and *false* itemsets that are frequent in the sample but not in the database. We claim that judicious modifications to simple random sampling can make sampling a viable means for attaining both high performance and acceptably accurate results.

This paper introduces FAST (Finding Associations from Sampled Transactions), a refined sampling-based algorithm that attempts to reduce the errors caused by sampling fluctuations. FAST is distinguished from previous sampling-based algorithms by its novel two-phase approach to sample collection. In Phase I, a large initial sample of transactions is collected and used to quickly and accurately estimate the support of each individual item in the database. In Phase II, a small final sample is obtained from the initial sample in such a manner that the support of each item in the final sample is as close as possible to the (estimated) support of the item in the entire database. We present two different approaches to obtaining the final sample in Phase II: "trimming" and "growing." The trimming procedure starts from the large initial sample and continues removing "outlier" transactions until a specified stopping criterion is satisfied. In contrast, the growing procedure selects "representative" transactions from the initial sample and adds them to an initially empty data set. In either approach, by forcing the supports of the single-item itemsets in the sample to approximate those in the database, the Phase II procedure helps ensure that the support of every itemset in the sample is close to that in the database. Such proximity in turn ensures that the set of association rules discovered in the sample overlaps to a high degree with the actual set of association rules. Indeed, our numerical experiments indicate that for any fixed computing budget, FAST identifies more frequent itemsets and fewer false itemsets than naive sampling-based algorithms. Moreover, because the expensive operation of identifying the association rules is performed on the (small) final sample, FAST can identify most, if not all, of the frequent itemsets in a database at an overall cost that is much lower than that of classical algorithms.

We emphasize that the sample created by FAST can be subsequently processed by *any* existing (non-sampling-based) association-rule algorithm, so that FAST complements, rather than replaces, current algorithms such as DepthMiner [2], Max-Miner [5], DIC [7], or FP-tree [14]. The FAST technique is especially compatible with algorithms — such as DepthMiner — that are designed for memory-resident data sets. Moreover, our new sampling technique can potentially be applied to render scalable other mining and statistical algorithms that use "count" data, such as clustering algorithms and contingency-table analysis.

2. SAMPLING AND MINING

Sampling is a powerful data reduction technique that has been applied to a variety of problems in database systems. Kivinen and Mannila [17] discuss the general applicability of sampling to data mining, and Zaki, et al. [22] employ a simple random sample to identify association rules.

Toivonen [20] uses sampling to generate candidate itemsets but still requires a full database scan. To minimize the number of missed frequent itemsets, the algorithm (1) lowers the minimum support when generating the candidate sets and (2) augments the initial collection C_0 of candidate sets with a "negative border." An itemset is a member of the negative border if all its subsets are elements of C_0. After the candidate itemsets are generated, they are tested by using the remainder of the database, i.e., the database minus the original sample. This testing is done in an effort to remove false itemsets.

John and Langley [16] give a dynamic sampling method that selects the sample size based on the observed behavior of the data-mining algorithm. Although dynamic sampling is developed in the context of a classification algorithm, the idea of "growing" a sample until a stopping criterion is satisfied is applicable in the setting of association-rule mining. Indeed, the approach in [16] originally motivated the FAST-grow algorithm presented here.

In the statistical literature, a well known technique for improving estimates obtained from a random sample is to "trim" the sample prior to computing the estimate. The idea is to make the sample more accurately reflect the properties of the parent population by removing "outlier" observations that are not sufficiently representative of the data set as a whole. Tukey's "trimmed mean" estimator — see, for example, [18, pp. 29–31] — is a classic example of this approach. The textbooks of Barnett and Lewis [6] and Huber [15] discuss this and other "robust estimation techniques." It is in this spirit that FAST creates a small final sample of transactions from a larger initial sample prior to mining, that is, prior to estimating the support of itemsets in the database from the support of the itemsets in the sample. In the classical setting of real-valued data, an observation is considered an outlier if it lies more than a specified distance away from the sample mean or median. It is not at all obvious, however, what an "outlier" should be in the context of association-rule mining. A key contribution of this paper is to provide a useful operational definition of "outlier" in the current nonclassical setting, together with an inexpensive technique for identifying such outliers. In the following section we describe our approach in detail.

3. THE FAST-TRIM ALGORITHM

3.1 Overview of the Algorithm

Throughout, we assume that the contents of the transac-

tional database do not change during the mining process. We also assume that the database is very large.

Denote by D the database of interest, by S a simple random sample drawn without replacement from D, and I the set of all items that appear in D. Also denote by $\mathcal{I}(D)$ the collection of itemsets that appear in D; a set of items A is an element of $\mathcal{I}(D)$ if and only if the items in A appear jointly in at least one transaction $t \in D$. Accordingly, the collection $\mathcal{I}(S)$ represents itemsets in S; of course, $\mathcal{I}(S) \subseteq \mathcal{I}(D)$. For $k \geq 1$ we denote by $\mathcal{I}_k(D)$ and $\mathcal{I}_k(S)$ the collection of k-itemsets in D and S, respectively. Similarly, $L(D)$ and $L(S)$ denote the frequent itemsets in D and S, and $L_k(D)$ and $L_k(S)$ the collection of frequent k-itemsets in D and S, respectively. For an itemset $A \subseteq I$ and a set of transactions T, let $n(A;T)$ be the number of transactions in T that contain A and let $|T|$ be the total number of transactions in T. Then the support of A in D and in S is given by $f(A;D) = n(A;D)/|D|$ and $f(A;S) = n(A;S)/|S|$, respectively.

Given a specified minimum support p and confidence c, the FAST-trim algorithm proceeds as follows:

1. Obtain a simple random sample S from D.
2. Compute $f(A;S)$ for each 1-itemset $A \in \mathcal{I}_1(S)$.
3. Using the supports computed in Step 2, obtain a reduced sample S_0 from S by trimming away outlier transactions as described in Section 3.3 below.
4. Run a standard association-rule algorithm against S_0 — with minimum support p and confidence c — to obtain the final set of association rules.

Olken [19] provides a review of techniques that can be used in Step 1 to obtain a random sample of transaction records. In general, the cost of obtaining a sample depends upon how the data is stored. In our implementation of FAST-trim, the transaction data is stored in a flat file and we use a sampling algorithm with a cost of $O(|S|)$ as in Ernvall and Nevalainen [9].

The computation in Step 2 of $f(A;S)$ for each 1-itemset $A \in \mathcal{I}_1(S)$ is straightforward: a count is maintained for each item present in S. If each of these items and the associated counts are stored together in a hash tree, then the cost of Step 2 is at most $O(T_{\max} \cdot |S|)$, where $T_{\max}$ stands for the maximal transaction length. Because the cost of Step 2 is relatively low, the sample S can be relatively large, thereby helping to ensure that the estimated supports are accurate.

The crux of the algorithm is Step 3, in which outlier transactions are trimmed from the sample. The following subsections discuss the choice of distance functions, the trimming procedure and the stopping criteria.

3.2 Distance Functions

As discussed previously, we define an outlier to be a transaction whose removal from the sample maximally reduces (or minimally increases) the discrepancy between the supports of the 1-itemsets in the sample and the corresponding supports in the database D. (Since the supports of the 1-itemsets in D are unknown, we estimate them by the corresponding supports in S as computed in Step 2 of FAST-trim.) To make the notion of "discrepancy" between 1-itemset supports precise we define a distance function, based on the symmetric set difference, by setting

$$\mathrm{Dist}_1(S_0, S) = \frac{|L_1(S) - L_1(S_0)| + |L_1(S_0) - L_1(S)|}{|L_1(S_0)| + |L_1(S)|} \quad (1)$$

for each subset $S_0 \subseteq S$ — in accordance with our previous notation, $L_1(S_0)$ and $L_1(S)$ denote the sets of frequent 1-itemsets in S_0 and S. Observe that $0 \leq \mathrm{Dist}_1 \leq 1$, and that Dist_1 is sensitive to both false frequent 1-itemsets and missed frequent 1-itemsets. Our goal is to trim away transactions from S so that the distance from the final sample S_0 to the initial sample S is as small as possible. We note that other definitions of distance are possible, for example

$$\mathrm{Dist}_2(S_0, S) = \sum_{A \in \mathcal{I}_1(S)} \big(f(A;S_0) - f(A;S)\big)^2, \quad (2)$$

$$\mathrm{Dist}_3(S_0, S) = \sum_{A \in \mathcal{I}_1(S)} |f(A;S_0) - f(A;S)|, \quad (3)$$

and

$$\mathrm{Dist}_4(S_0, S) = \max_{A \in \mathcal{I}_1(S)} |f(A;S_0) - f(A;S)|. \quad (4)$$

Observe that Dist_2, Dist_3, and Dist_4 correspond to L^p-norm distances between $f(\,\cdot\,;S_0)$ and $f(\,\cdot\,;S)$ with $p = 2$, 1, and ∞, respectively. Because of the well-known equivalence between these norms, we expect algorithms based on Dist_2, Dist_3, and Dist_4 to behave similarly, and we focus on differences between Dist_1 and Dist_2.

3.3 Trimming the Sample

Suppose at first that the goal is to produce a final sample S_0 containing exactly n (≥ 1) transactions — note that the value of n is directly related to the time subsequently required to generate the frequent itemsets. Given an initial sample S, we therefore wish to find a solution S_0 to the following problem:

$$\underset{S_0 \subseteq S,\ |S_0| = n}{\text{minimize}} \ \mathrm{Dist}(S_0, S). \quad (5)$$

This combinatorial optimization problem is extremely expensive to solve exactly. Indeed, it can be shown that the problem is NP-complete, by reduction from the One-In-Three SAT problem [12]. There are, however, a variety of heuristic algorithms that yield approximate solutions to the problem in (5).

A trivial algorithm is to trim "obliviously:" initially set $S_0 = S$ and then scan the transactions in S_0 in an arbitrary order, removing each transaction in turn until $|S_0| = n$. Although this procedure is inexpensive — e.g., no evaluations of Dist() are required — it is clear that the final sample S_0 will not be any more representative of the database D than the initial sample S.

An alternative greedy algorithm also starts by setting $S_0 = S$, and then proceeds in stages. At each stage the algorithm finds a transaction $t^* \in S_0$ such that $\mathrm{Dist}(S_0 - \{t^*\}, S) \leq \mathrm{Dist}(S_0 - \{t\}, S)$ for $t \in S_0$ and sets $S_0 = S_0 - \{t^*\}$. Although this algorithm produces much better solutions than the "oblivious" algorithm — each transaction that is removed is known to be at least as much of an outlier as any other transaction currently in S_0 — it is prohibitively expensive: when the current sample contains j ($> n$) transactions, precisely j evaluations of Dist() are required to remove the next transaction. The total number of Dist() evaluations required to produce the final sample S_0 is

```
obtain a simple random sample S from D;
compute f(A; S) for each A ∈ I_1(S);
set S_0 := S;
while (|S_0| > n) {  //trimming phase
    divide S_0 into disjoint groups of min(k, |S_0|)
        transactions each;
    for each group G {
        compute f(A; S_0) for each A ∈ I_1(S_0);
        set S_0 = S_0 − {t*}, where Dist(S_0 − {t*}, S) =
              min_{t∈G} Dist(S_0 − {t}, S);
    }
}
repeat {  //auxiliary trimming phase
    divide S_0 into disjoint groups of min(k, |S_0|)
        transactions each;
    for each group G {
        if (∃t* such that Dist(S_0 − {t*}, S) =
          min_{t∈G} Dist(S_0 − {t}, S) and
          Dist(S_0 − {t*}, S) ≤ Dist(S_0, S)) {
            set S_0 = S_0 − {t*};
            compute f(A; S_0) for each A ∈ I_1(S);
        }
    }
}
until (no transaction t* is removed from S_0 for any group G);
run a standard association-rule algorithm against S_0 to
obtain final set of association rules;
```

Figure 1: Complete FAST-trim Algorithm

therefore $O(m^2-n^2)$, where m is the number of transactions in the initial sample S.

A hybrid algorithm uses an input parameter $k \in \{1, 2, \ldots, |S|\}$ to explicitly trade off speed and accuracy. As above, the algorithm initially sets $S_0 = S$. At each stage, the algorithm examines the transactions in S_0 in disjoint groups of size $\min(k, |S_0|)$. For each group, we select a transaction t that minimizes the function $\text{Dist}(S_0-\{t\}, S)$ over all transactions in the group and remove t from S_0. Observe that for $k = |S|$ the algorithm reduces to the greedy algorithm, whereas for k close to 1 the algorithm approximates the "oblivious" algorithm. Intermediate values of k may be chosen to trade off speed and accuracy: the larger the k value, the higher the accuracy, but the slower the speed. As before, a rough idea of the complexity can be obtained by considering the number of Dist() evaluations required to trim the sample. If $k \le n$, then roughly k evaluations of Dist() are required for each of the $(m-n)$ transactions removed from S_0, for a total of about $k(m-n)$ evaluations. If $k > n$, then $k(m-k)$ evaluations are required to remove the first $m-k$ transitions; from this point on, the algorithm behaves like the greedy algorithm, and (k^2-n^2) evaluations of Dist() are required to complete the trimming phase. It follows that the overall complexity is $O(km-n^2)$ if $k > n$ and $O(km-kn)$ if $k \le n$. We use this hybrid greedy heuristic in our implementation of FAST-trim.

3.4 Stopping Criteria

As formulated so far, the trimming procedure stops when the sample size reaches a specified value n. Note that after the desired final sample size is reached, additional trimming may further reduce the processing time without decreasing the accuracy much. In light of this observation, we add an auxiliary trimming phase to FAST that uses a distance-based stopping criterion. Specifically, the auxiliary trimming phase stops when, for each group during the current iteration, the removal of any transaction from the group will increase $\text{Dist}(S_0, S)$. In Section 5 we present experimental results for the FAST-trim algorithm both with and without the auxiliary trimming phase. The complete FAST-trim algorithm is presented in Figure 1.

4. OTHER VARIANTS OF FAST

The FAST-trim algorithm obtains the final sample S_0 by sequentially removing transactions from the initial sample S. Other approaches to obtaining S_0 from S lead to alternative versions of FAST.

4.1 The Growing-based FAST Algorithm

As an alternative to the trimming procedure described in Section 3, the exclusion of "outlier" transactions can be accomplished by selecting "representative" transactions from the original sample S and adding them to a second sample S_0 that is initially empty. In this section we discuss the resulting FAST-grow algorithm.

First consider a version of FAST-grow with a specified final sample size of n transactions. Like FAST-trim, the FAST-grow algorithm has an input parameter $k \in \{1, 2, \ldots, |S|\}$ and proceeds in stages. Initially, S_0 is empty. At each stage, FAST-grow increments S_0 by adding representative transactions. In order to identify representative transactions, the transactions in $S - S_0$ are divided into disjoint groups, with each group having $\min(|S - S_0|, k)$ transactions. For each group, the algorithm selects a transaction t^* that minimizes the function $\text{Dist}(S_0 \cup \{t\}, S)$ over all transactions in the group and adds t^* to S_0. The algorithm proceeds until $|S_0| = n$. As with the FAST-trim algorithm, k is chosen to trade off speed and accuracy: the larger the k value, the higher the accuracy, but the slower the speed. To quantify the complexity in terms of the number of Dist() evaluations required, let m be, as before, the number of transactions in the initial sample S. Then an argument analogous to that given for FAST-trim shows that the total number of Dist() evaluations required to create the final sample is $O\big(km-(m-n)^2\big)$ if $k > m-n$ and $O(nk)$ if $k \le m-n$.

As with FAST-trim alternative stopping criteria are available for FAST-grow. Denote by $S_0(i)$ the transactions in S_0 at the end of stage i of the algorithm, and by K the index of the final stage — i.e., $|S_0(K)| = n$. It may be the case that $\text{Dist}(S_0(i), S) \le \text{Dist}(S_0(K), S)$ for some $i < K$. If so, then we can achieve acceptable accuracy by running the association-rule algorithm against $S_0(i)$ rather than $S_0(K)$. Since $|S_0(i)| < |S_0(K)|$ by definition of FAST-grow, the overall time to mine the association rules will usually be shorter. Indeed, the optimal accuracy-maximizing strategy is to run the association-rule algorithm against the set $S_0(i^*)$, where

$$i^* = \min\Big\{\, 1 \le i \le K:\ \ \text{Dist}\big(S_0(i), S\big) = \min_{1 \le j \le K} \text{Dist}\big(S_0(j), S\big)\ \Big\}.$$

In other words, a rollback operation is applied to obtain the smallest sample that has the potentially highest final accuracy. This rollback operation can be viewed as a rough analogue of the auxiliary-trimming phase in the FAST-trim algorithm. Other variations on the stopping criteria for FAST-trim and FAST-grow are possible — our experiments indicate, however, that the simplest stopping criteria are best, so we do not investigate these variations in detail.

```
obtain a simple random sample S from D;
compute f(A; S) for each A ∈ I_1(S);
set i := 0, S_0(i) := ∅, minDist := ∞, and minStage := −1;
while (|S_0(i)| < n) {
    divide S − S_0(i) into disjoint groups of min(|S − S_0(i)|, k)
            transactions each;
    for each group G {
        set S_0(i) := S_0(i) ∪ {t*}, where Dist(S_0(i) ∪ {t*}, S) =
                                    min_{t∈G} Dist(S_0(i) ∪ {t}, S);
    }
    compute f(A; S_0(i)) for each A ∈ I_1(S)
    if (Dist(S_0(i), S) < minDist) {
        set minDist := dist(S_0(i), S) and minStage := i;
    }
    set S_0(i + 1) := S_0(i);
}
run a standard association-rule algorithm against
S_0(minStage) to obtain final set of association rules;
```

Figure 2: Complete FAST-grow Algorithm

The complete FAST-grow algorithm is illustrated in Figure 2. Of course, we do not separately store the sets $S_0(0)$, $S_0(1)$, ..., $S_0(K)$ — we merely keep track of the stage at which each transaction is added to S_0. This information is all that is needed for rollback.

4.2 Randomized Versions of FAST

Another approach to solving the problem in (5) is to use randomized algorithms, such as random swapping, simulated annealing, tabu search, and genetic algorithms. These algorithms did not perform well relative to our heuristics, and so we leave a more comprehensive investigation of randomized algorithms as a topic for future research.

5. PERFORMANCE STUDY

In this section we describe an empirical study carried out to evaluate the performance of FAST. All experiments were performed on an HP 9000 series UNIX multi-user workstation with a processor speed of 132 MHz.

5.1 Experimental Methodology

We used both a synthetic and a real-world database. The synthetic database was generated using code from the IBM QUEST project [4]. The parameter settings for synthetic data generation are similar to those in [4]: the total numbers of items and transactions are set to 1000 and 100,000, the number of maximal potentially frequent itemsets is 2000, the average length of transactions is 10, and the average length of maximal potentially large itemsets is 4. We used a minimum support of 0.77%, at which level there are neither too many nor too few large itemsets, and the length of the maximal large itemsets is 6. The real-world database is a sales database from a large retailing company, and is similar to the one used in [3]. We obtained similar results on both databases and therefore focus on the synthetic database. In a certain sense, the synthetic databases pose more of a challenge to the FAST algorithms than the real-world data. This is because the synthetic data contains relatively many frequent k-itemsets with $k > 1$, whereas the trimming and growing heuristics are based on 1-itemset frequencies—most of the frequent itemsets in the real-world database are 1-itemsets.

We performed experiments using FAST-trim, FAST-grow, Toivonen's Algorithm, and "SRS-Apriori", which is simple random sampling combined with the Apriori algorithm as in [22]. Distance functions Dist_1 and Dist_2 are used in FAST-trim and FAST-grow. Preliminary experiments showed that a value of $k = 10$ for the group-size parameter worked well in both FAST-trim and FAST-grow, and we therefore use this value throughout. We use a sampling ratio of 30% throughout to create the initial sample for FAST and its variants. We also use the Apriori algorithm to obtain frequent itemsets in the final step of the FAST algorithms — this choice permits fair comparisons with Toivonen's Algorithm and SRS-Apriori.

For each database and sampling-based mining algorithm, we executed the algorithm 50 times, each time choosing a different simple random sample (without replacement) from the database. Thus each performance number reported below is an average of 50 observations. Our primary measure of accuracy is as follows:

$$accuracy = 1 - \frac{|L(D) - L(S)| + |L(S) - L(D)|}{|L(D)| + |L(S)|}, \qquad (6)$$

where, as before, $L(D)$ and $L(S)$ denote the frequent itemsets from the database D and the sample S. As with the distance measure, the accuracy measure in (6) is based on the symmetric set difference, and hence is sensitive to both false and missing frequent itemsets.

5.2 Experimental Results

In this section we examine the performance of the different algorithms in terms of accuracy and execution time. For FAST-trim, we consider both the "fixed-size" stopping criterion (no auxiliary trimming phase) and "min-distance" stopping criterion (which uses the auxiliary trimming phase). We similarly consider fixed-size and min-distance stopping criteria for FAST-grow. The results are presented using an abbreviated notation in which, for example, FAST-t-D1 denotes the FAST-trim algorithm based on Dist_1 and FAST-g-D2 denotes the FAST-grow algorithm based on Dist_2. The final sampling ratios chosen are 5%, 7.5%, 10%, 12.5% and 15%. For SRS-Apriori, two additional sampling ratios of 20% and 30% are also selected.

5.2.1 Accuracy vs. Sampling Ratio

Figures 3 and 4 illustrate the accuracy of the different algorithms on the synthetic databases. As shown in the figures, FAST-trim and FAST-grow outperform SRS-Apriori in most cases. Indeed, all fixed-size versions of FAST are more accurate than SRS-Apriori, especially those that use Dist_2—with a 5% sample, these latter algorithms achieve results comparable to a 15% simple random sample. Most FAST-trim and FAST-grow algorithms missed between 3.5–6% of the frequent itemsets at a final sample size of 5%, compared with 11.57% for a 5% simple random sample and 6% for a 20% simple random sample. Moreover, the FAST algorithms typically generate 30–50% fewer false itemsets than does SRS-Apriori. The metric Dist_2 seems to be more effective than Dist_1 in most cases. One possible reason for this is that Dist_1, unlike Dist_2, does not penalize for a poor approximation of a 1-itemset frequency if this frequency is less than the minimum support. Thus the entire set of 1-itemset frequencies may not be approximated as well in a

global sense, and consequently the k-itemset frequencies are not approximated to sufficient accuracy.

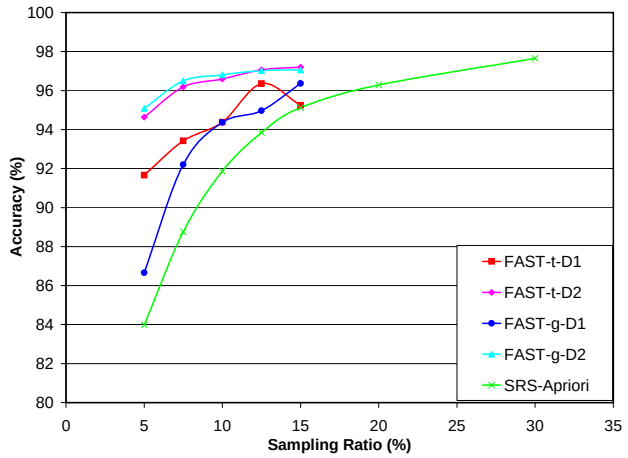

Figure 3: Accuracy vs. Sampling Ratio on Synthetic Data w/ Fixed-Size Stopping Criterion

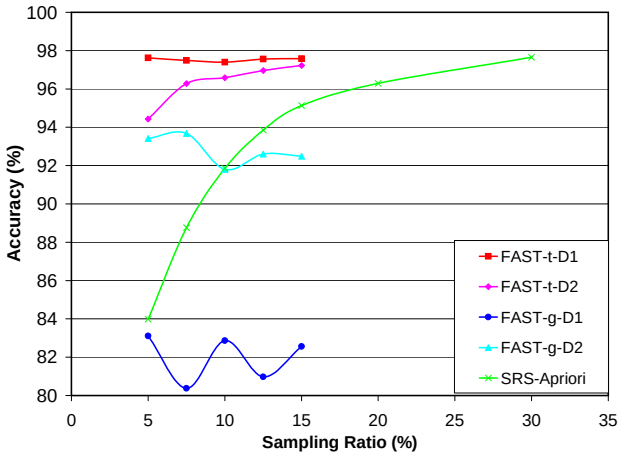

Figure 4: Accuracy vs. Sampling Ratio on Synthetic Data w/ Min-Distance Stopping Criterion

The FAST-trim algorithms appear to benefit from the min-distance stopping criterion. For example, FAST-t-D1 quickly achieves a high level of accuracy — a 5% final sample size gives results comparable to a 30% simple random sample. In contrast, the performance of the FAST-grow algorithms degrades under the min-distance stopping criterion. The problem appears to be that the final sample size is too small. For example, after FAST-g-D1 executes the rollback operation, the final sample size is typically equal to about 0.2%. Although there are few false or missing large 1-itemsets in the small sample, a large number of false k-itemsets are generated for $k > 1$. The problems associated with using Dist_1 are magnified in this situation, which is why the performance of FAST-g-D1 is particularly bad.

5.2.2 *Execution Time vs. Sampling Ratio*

Figure 5 displays the total execution time (sampling plus subsequent frequent-itemset generation) of both SRS-Apriori and various FAST algorithms on the synthetic database as a function of the sampling ratio. We consider only FAST algorithms that use the fixed-size stopping criterion — as discussed in the previous section, use of this criterion results in better performance and more stable behavior. As shown in the figure, the execution times of the FAST algorithms were all very similar to the corresponding times for SRS-Apriori. Thus the additional processing time — relative to simple random sampling — that is required by FAST to trim or grow a sample is insignificant compared to the time required to generate the frequent itemsets.

5.2.3 *Accuracy vs. Execution Time*

Our "bottom-line" performance results are displayed in Figure 6, which illustrates the tradeoffs between accuracy and execution time for various FAST algorithms and for SRS-Apriori. As shown in the figure, almost all FAST variations performed better or much better than SRS-Apriori.

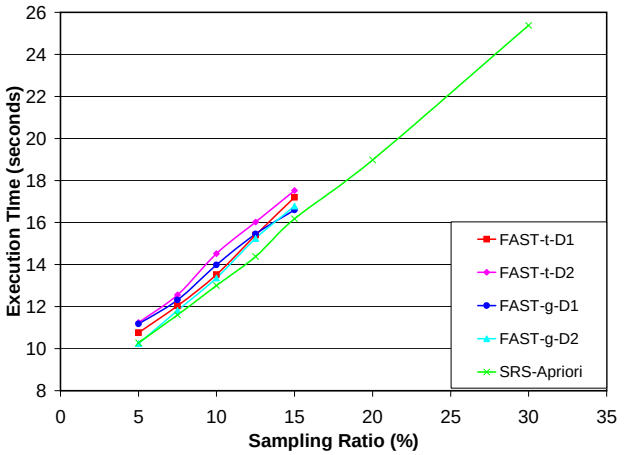

Figure 5: Execution Time vs. Sampling Ratio on Synthetic Data w/ Fixed-Size Stopping Criterion

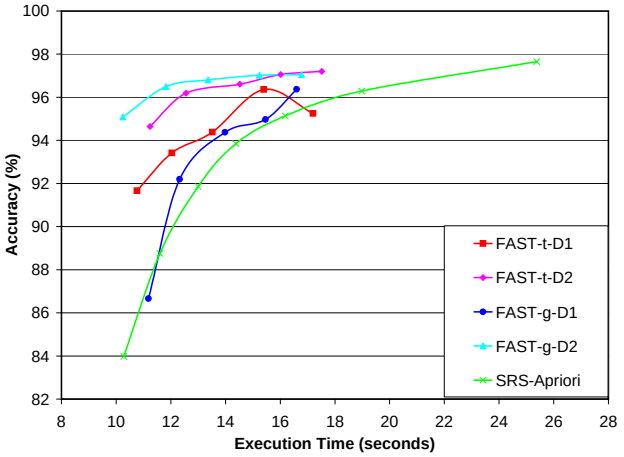

Figure 6: Accuracy vs. Execution Time on Synthetic Data w/ Fixed-Size Stopping Criterion

Specifically, if required to finish the mining task within ten seconds, FAST-g-D2 can achieve an accuracy of approximately 94%, compared to an accuracy of 84% for SRS-Apriori. Looked at another way, FAST-g-D2 achieves an accuracy comparable to SRS-Apriori in about 35% less time.

We also can compare the performance of FAST with that of Toivonen's Algorithm. Within 10 seconds, FAST-g-D2 can process a 5% final sample and achieve approximately a 95% accuracy. On the contrary, it takes about 100 seconds, or about 10 times longer, for Toivonen's algorithm to finish the mining task using the same sample, whereas its resulting accuracy is 99.8%. The problem with Toivonen's Algorithm is that even though the generation of frequent itemsets from the sample is relatively inexpensive, the pass over the remaining database to eliminate false itemsets can be quite expensive. The expense is especially high when the database contains a large number of long transactions and/or long frequent itemsets. This problem persists no matter how small the sample is — note that the smaller the sample, the larger the remaining database. Also, unlike with FAST, the user has no real control over the tradeoff between speed and accuracy: neither the lowered minimum support value nor the size of the negative border can be manipulated by the user. The discrepancies in processing time between FAST and Toivonen's algorithm only increase as the database becomes larger. Thus if extremely high accuracy is of paramount importance and processing time is not an issue, then Toivonen's algorithm is a reasonable choice. Otherwise, FAST is clearly the algorithm of choice.

6. CONCLUSIONS AND FUTURE WORK

We have introduced in this paper an efficient two-step data reduction approach based on sampling and tailored to the mining of count data, in particular, the fast discovery of association rules. In the first step, a relatively large simple random sample is obtained and used to estimate the support of each item in the database. In the second step, a final small sample is created either by trimming outliers or selecting representative transactions based on a distance function that incorporates the 1-itemset supports computed in the first step.

An empirical study using both a real database and a synthetic database supports our claims of efficiency and accuracy. FAST was able to achieve 90-95% accuracy using a final sample size 15–33% as large as the simple random sample used by SRS-Apriori. This efficiency gain resulted in a speedup by roughly a factor of 10 over algorithms, even efficient ones such as that of Toivonen, that require one or more expensive passes over the entire database. Unlike the latter algorithms, the user of FAST has relatively fine control, by means of the adjustable algorithm parameters k and n, over the tradeoff between speed and accuracy.

As mentioned in the text, the detailed issues involved in combining the FAST sampling technique with some of the more recent association-rule algorithms, such as Apriori, DIC, Max-Miner, DepthMiner, or FP-tree, need to be investigated. In general, we plan on exploring the efficacy of our sampling technique for other mining and statistical analysis tasks for count data. It would be desirable to push the ideas developed in this paper even further, perhaps combining them with online processing ideas as in [11] or [13], in order to make data mining systems even more interactive and subject to user control.

7. ACKNOWLEDGEMENT

We would like to thank Rakesh Agrawal, Roberto Bayardo, Lisa Hellerstein, Herve Bronimann, Lisa Singh, and R. Srikant for helpful discussions.

8. REFERENCES

[1] S. Acharya, P. B. Gibbons, and V. Poosala. Congressional samples for approximate answering of group-by queries. In *Proc. Proc. 2000 ACM SIGMOD*, 2000.

[2] R. C. Agarwal, C. C. Aggarwal, and V. V. V. Prasad. Depth first generation of long patterns. In *Proc. Sixth ACM SIGKDD*, 2000.

[3] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proc. 1993 ACM SIGMOD*, 1993.

[4] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. 20th VLDB*, 1994.

[5] R. J. Bayardo. Efficiently mining large patterns from databases. In *Proc. 1998 ACM SIGMOD*, 1998.

[6] V. Barnett and T. Lewis. Outliers in Statistical Data. *John Wiley & Sons*, New York, 1994.

[7] S. Brin, R. Motwani, J. Ullman, and S. Tsur. Dynamic itemset counting and implication rules for market basket data. In *Proc. 1997 ACM SIGMOD*, 1997.

[8] S. Choudhuri, M. Datar, R. Motwani, V. Narasayya. Overcoming limitations of sampling for aggregation queries. In *Proc. 17th ICDE*, 2001.

[9] J. Ernvall and O. Nevalainen. An algorithm for unbiased random sampling. *Comput. J.*, 25:45–47, 1982.

[10] P. Gibbons. Distinct sampling for highly-accurate answers to distinct values queries and event reports. In *Proc. VLDB Conference.*, 2001.

[11] P. J. Haas. Techniques for Online Exploration of Large Object-Relational Datasets. In *Proc. 11th Intl. Conf. Scientific and Statistical Database Management*, pages 4–12. IEEE Press, 1999.

[12] Lisa Hellerstein. Proof of the NP-completeness of FAST. *Private Correspondence*, 2000.

[13] C. Hidber. Online association rule mining. In *Proc. ACM SIGMOD*, 1999.

[14] Jiawei Han, Jian Pei and Yiwen Yin. Mining frequent patterns without candidate generation. In *Proc. ACM SIGMOD*, 2000.

[15] P. J. Huber. *Robust Statistics*. Wiley, New York, 1981.

[16] G. H. John and P. Langley. Static versus dynamic sampling for data mining. In *Proc. Second Intl. Conf. Knowledge Discovery and Data Mining*, pages 367–370. AAAI Press, 1996.

[17] J. Kivinen and H. Mannila. The power of sampling in knowledge discovery. In *Proc. Thirteenth ACM SIGACT-SIGMOD-SIGART Symp. Principles of Database Sys.*, pages 77–85. ACM Press, 1994.

[18] R. G. Miller. *Beyond ANOVA: Basics of Applied Statistics*. Wiley, New York, 1986.

[19] F. Olken. *Random Sampling from Databases*. Ph.D. Dissertation, University of California, Berkeley, CA, 1993. Available as Tech. Report LBL-32883, Lawrence Berkeley Laboratories, Berkeley, CA.

[20] H. Toivonen. Sampling large databases for association rules. In *Proc. 22nd VLDB*, 1996.

[21] R. Winter and K. Auerbach. The big time: 1998 Winter VLDB survey. *Database Programming Design*, August, 1998.

[22] M. J. Zaki, S. Parthasarathy, W. Lin, and M. Ogihara. Evaluation of sampling for data mining of association rules. Technical Report 617, University of Rochester, Rochester, NY, 1996.