Ef£cient Data Reduction with EASE

Hervé Brönnimann*
Comp & Info Sci
Polytechnic Univ.
Brooklyn, NY
hbr@poly.edu

Bin Chen
Exelixis Inc.
San Francisco, CA
bchen@ece.nwu.edu

Manoranjan Dash†
Elect & Comp Engg
Evanston, IL
Northwestern Univ.
manoranj@ece.nwu.edu

Peter Haas
IBM Almaden
San Jose, CA
peterh@almaden.ibm.com

Peter Scheuermann‡
Elect & Comp Engg
Evanston, IL
Northwestern Univ.
peters@ece.nwu.edu

ABSTRACT

A variety of mining and analysis problems — ranging from association-rule discovery to contingency table analysis to materialization of certain approximate datacubes — involve the extraction of knowledge from a set of categorical count data. Such data can be viewed as a collection of "transactions," where a transaction is a fixed-length vector of counts. Classical algorithms for solving count-data problems require one or more computationally intensive passes over the entire database and can be prohibitively slow. One effective method for dealing with this ever-worsening scalability problem is to run the algorithms on a small sample of the data. We present a new data-reduction algorithm, called EASE, for producing such a sample. Like the FAST algorithm introduced by Chen et al., EASE is especially designed for count data applications. Both EASE and FAST take a relatively large initial random sample and then deterministically produce a subsample whose "distance" — appropriately defined — from the complete database is minimal. Unlike FAST, which obtains the final subsample by quasi-greedy descent, EASE uses epsilon-approximation methods to obtain the final subsample by a process of repeated halving. Experiments both in the context of association rule mining and classical χ^2 contingency-table analysis show that EASE outperforms both FAST and simple random sampling, sometimes dramatically.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications - Data Mining

*Research of this author has been partially supported by NSF CAREER Grant CCR-0133599.

†Research of this author has been partially supported by NSF Grant EIA-0000536.

‡Research of this author has been partially supported by NSF Grant EIA-0000536.

SIGKDD '03, August 24-27, 2003, Washington, DC, USA

Keywords

Sampling, frequency estimation, count dataset, association rules, OLAP, data streams

1. INTRODUCTION

A variety of important knowledge-extraction problems take as their input a set of categorical count data. Conceptually, such a data set can be viewed as a "datacube" in the sense of [8], where the "measure" in each cell is a frequency count. In the context of association-rule mining, for example, each dimension corresponds to one of a finite set of items. The two possible values along each dimension are 1 and 0, corresponding to the presence or absence of the item in a user transaction. The highest-level aggregates — after the sum over all cells in the cube — correspond to the (unnormalized) "1-itemset supports." Sets of categorical count data are called "contingency tables" in the setting of classical statistical analysis, and the high-level aggregates are called "marginal frequencies." Such data sets also serve as the input to an important class of OLAP query processing tasks. In practice, the raw data for the foregoing mining and analysis algorithms is often stored in a database as a set of transactions, where each transaction can be viewed as a fixed-length vector of counts. E.g., in the association rule setting, a transaction can be viewed as a vector whose length is equal to the number of items; a 1 (resp., 0) in the jth position corresponds to the presence (resp., absence) of item j in the transaction. Similarly, for a $K \times N$ contingency table, a "transaction" can be viewed as a vector of length KN, where each position in the vector represents one of the cells in the table; exactly one of the components is equal to 1, and the remaining components are equal to 0. Of course, in real systems other data encodings may actually be used.

Real-world processing of categorical count data is becoming increasingly difficult, due to ever-increasing volumes of data in warehouses and on the web. Mining and analysis algorithms that require one or more computationally intensive passes over the entire database are becoming prohibitively slow, and this problem will only become worse in the future. One means of addressing scalability dilemma is to run the algorithms on a sample of the data. Use of a simple random sample, however, can lead to unsatisfactory results. The problem is that such a sample may not adequately represent the entire data set due to random fluctuations in the sampling process. This difficulty is particularly apparent at small sample sizes.

In this paper we introduce a novel data-reduction method, called EASE (*E*psilon *A*pproximation: *S*ampling *E*nabled), that is especially designed for categorical count data.[1] This algorithm is an outgrowth of earlier work by Chen, et al. on the FAST data-reduction method [7]. Both EASE and FAST start with a relatively large simple random sample of transactions and deterministically trim the sample to create a final subsample whose "distance" from the complete database is as small as possible. For reasons of computational efficiency, both algorithms define the subsample as "close" to the original database if the high-level aggregates of the subsample — normalized by the total number of data points — are "close" to the normalized aggregates in the database. These normalized aggregates typically correspond to 1-itemset or 2-itemset supports in the association-rule setting or, in the setting of a contingency table, relative marginal or cell frequencies. We actually use the normalized high-level aggregates in the initial sample — which can be calculated quickly and easily — as surrogates for those in the original database.

The key innovation of EASE lies in the method by which the final subsample is obtained. Unlike FAST, which obtains the final subsample by trimming away outliers in a process of quasi-greedy descent, EASE uses ε-approximation methods to obtain the final subsample by a process of repeated halving. Unlike FAST, the EASE algorithm provides a guaranteed upper bound on the distance between the initial sample and final subsample. In addition, EASE can process transactions on the fly, i.e., a transaction needs to be examined only once to determine whether it belongs to the final subsample. Moreover, the average time needed to process a transaction is proportional to the number of positive components in that transaction. We show that EASE leads to much better estimation of frequencies than SRS. Preliminary experiments in the context of both association-rule mining and classical contingency-table analysis indicate that EASE outperforms both FAST and SRS.

Our approach is similar in spirit to the ε-deficient synopsis of [10]. The main difference is that our goal is to produce a sample that can be used in multiple ways, whereas the algorithms in [10] focus on producing a more specialized synopsis of counts. Note that, given a minimum support $s > 0$, frequent items can be selected (conservatively) by identifying the items whose support on the subsample is at least $s-\varepsilon$. From a practical point of view, one should choose a value $\varepsilon \ll s$ to deploy the EASE algorithm. Our method provides a (2ε)-deficient synopsis of the data. Of course, the footprint of our method is much larger, but in addition to the synopsis, we also obtain a sample of the data.

The paper is organized as follows. In Section 2 we introduce the new EASE algorithm, and the FAST algorithm is briefly reviewed in Section 3. Section 4 discusses some practical considerations when selecting the initial random sample. Sections 5 presents a short study on approximating frequencies, and Sections 6 and 7 present two case studies, one on mining association rules and the other on testing statistical independence using a classical contingency-table analysis. Section 8 concludes our paper with some suggestions for further research.

[1]A preliminary, relatively inefficient version of the algorithm, called multi-pass EA, was originally introduced in [5]. The version of the algorithm presented there requires multiple passes through the data and does not exploit sampling.

Notation. For ease of exposition, we describe both FAST and EASE using language and notation appropriate to the context of association-rule mining; in other settings, the "mapping" from the mining context to the application at hand is usually straightforward. Denote by D the database of interest, by S a simple random sample drawn without replacement from D, and by I the set of all items that appear in D. Let $N = |D|$ be the number of transactions in D, $n = |S|$ be the number of transactions in S, and $m = |I|$ be the number of items. 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 $t \in D$. If A contains exactly k (≥ 1) elements, then A is sometimes called a k-itemset. In particular, the 1-itemsets are simply the original items. The collection $\mathcal{I}(S)$ denotes the itemsets that appear 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.

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 threshold $s > 0$, an item is *frequent* in D (resp., in S) if its support in D (resp., in S) is no less than s. We denote by $L(D)$ and $L(S)$ 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.

2. EPSILON APPROXIMATION METHOD

The epsilon approximation method tries to find a small subset having 1-itemset supports that are close to those in the entire database. The *discrepancy* of any subset S_0 of a superset S (that is, the distance between S_0 and S with respect to the 1-itemset frequencies) is computed by using the $Dist_\infty$ metric:

$$Dist_\infty(S_0, S) = \max_{A \in \mathcal{I}_1(S)} \Big| f(A;S_0) - f(A;S) \Big| \tag{1}$$

The sample S_0 is called an ε-approximation of S if its discrepancy is bounded by ε. Obviously there is a trade-off between the size of the sample and ε: the smaller the sample, the larger the value of ε.

The concept of ε-approximation has been widely used in machine learning [13] and derandomization of geometric algorithms [4, 11]. A seminal result of Vapnik and Chervonenkis shows that a random sample of size $O(\frac{d}{\varepsilon^2} \log \frac{1}{\varepsilon})$, where d is the "VC dimension" (assumed finite) of the dataset, is an ε-approximation. This result establishes the link between random samples and frequency estimations over several items simultaneously. For literature on ε-approximations, see e.g. the books by Alon and Spencer [3] Chazelle [6, Ch.4].

Given a specified $\varepsilon > 0$, the EASE algorithm proceeds sequentially as follows:

1. Obtain a simple random sample S from D by sampling each transaction online with probability $|S|/|D|$.
2. For each transaction in S, decide whether to keep it in S_0, maintaining the property that at any point S_0 is an ε-approximation for the set of currently seen transactions in S.

If we use Bernoulli sampling instead of simple random sampling (see Section 4 below), then both steps can be performed online, making EASE an online algorithm.

2.1 The halving method

The crux of the EASE algorithm is obviously Step 2. At the heart of the algorithm is a method that computes a subset of S approximately half the size which has good approximation properties. This method follows the hyperbolic cosine method (see e.g. Alon and Spencer's book [3, Ch.15]). We explain both this method and a variant originally due to Chazelle and Matoušek.

Specifically, denote by S^i the set of all transactions in S that contain item A_i. At the beginning, all transactions are grey (i.e. uncolored). For each transaction examined in Step 2, we make a choice of coloring the transaction red or blue. At the end, the subset S_r of S we retain consists of the red elements. In the next section, we explain how to compound these halving steps to result in a small subset S_0 such that $Dist_\infty(S_0, S) \leq \varepsilon$.

The hard part is of course to decide how to color an element. A random coloring will have the guarantees only with a certain probability. In order to obtain a deterministic algorithm (and actually a much better sample), for each item A_i we define a penalty Q_i (initially equal to 2). These penalties will be functions of r_i and b_i, the number of red and blue transactions in $S^i = A_i \cap S$. After coloring the first j transactions, these number will be $r_i = r_i^{(j)}$ and $b_i = b_i^{(j)}$, and the penalties will have some value $Q^{(j)}$. Since ideally we would like $r_i = b_i$, the penalty must be low for small $|r_i - b_i|$ and will increase exponentially in $|r_i - b_i|$. We choose penalties Q_i given by

$$Q_i = Q_i^{(j)} = Q_{i,1}^{(j)} + Q_{i,2}^{(j)}, \tag{2}$$

$$Q_{i,1}^{(j)} = (1+\delta_i)^{r_i}(1-\delta_i)^{b_i}, \tag{3}$$

$$Q_{i,2}^{(j)} = (1-\delta_i)^{r_i}(1+\delta_i)^{b_i}. \tag{4}$$

This penalty function has the shape depicted in Figure 1, and δ_i controls how steeply the penalty increases away from the equilibirum $r_i = b_i$. It is not hard to compute how the penalties need to be updated if we color the $(j+1)$-th transaction red (resp. blue). For a color $c = r, b$, denote by $Q_i^{(j\|c)}$ the i-th penalty given the coloring of the first j transactions, and that the $(j+1)$-th transaction is colored c; then:

$$Q_{i,1}^{(j\|r)} = (1+\delta_i)Q_{i,1}^{(j)}, \qquad Q_{i,2}^{(j\|r)} = (1-\delta_i)Q_{i,2}^{(j)},$$
$$Q_{i,1}^{(j\|b)} = (1-\delta_i)Q_{i,1}^{(j)}, \qquad Q_{i,2}^{(j\|b)} = (1+\delta_i)Q_{i,2}^{(j)}.$$

The decision whether to color an element red or blue is then very simple: choose the color c for which $Q^{(j\|c)} = \sum_i Q_i^{(j\|c)}$ is the smallest.

An important observation is that $Q_i^{(j\|r)} + Q_i^{(j\|b)} = 2Q_i^{(j)}$ for every item i, and so when summing over all items, by linearity $Q^{(j\|r)} + Q^{(j\|b)} = 2Q^{(j)}$. It follows that

$$\min\Big(Q^{(j\|r)}, Q^{(j\|b)}\Big) \leq \frac{1}{2}\Big(Q^{(j\|r)} + Q^{(j\|b)}\Big) = Q^{(j)}.$$

This means that the coloring chosen by the algorithm *never* increases the penalty!

By the symmetry of the penalty function, and since $r_i + b_i \leq |S^i|$, we have

$$Q_i^{(\text{final})} \geq (1+\delta_i)^{|r_i - b_i|}(1-\delta_i^2)^{|S^i|}$$

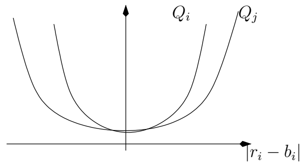

Figure 1: The penalty functions for the halving method. Depicted in the figure is how the penalty changes as a function of the size of the item, for two items i and j such that $|A_i| \ll |A_j|$.

From this and the fact that $Q^{(\text{init})} = 2m$, we get that the final overall penalty is at most $2m$, hence $Q_i^{(\text{final})} \leq 2m$ for every i, which in turns implies that [5]

$$|r_i - b_i| \leq \frac{\ln(2m)}{\ln(1+\delta_i)} + \frac{|S^i| \ln(1/(1-\delta_i)^2)}{\ln(1+\delta_i)}.$$

We can choose the value of δ_i to make the right-hand side as small as possible. The first (resp., second) term in the sum is decreasing (resp. increasing) in δ_i, and so a reasonable choice is to balance the two terms, leading to

$$\delta_i = \sqrt{1 - \exp\Big(-\frac{\ln(2m)}{|S^i|}\Big)}. \tag{5}$$

For that value, it can be shown that the final error on item i is at most $O(\sqrt{|S_i| \log(2m)})$ which, if S^i is not too small, is much smaller than $|S^i|$.

Since we always choose the red transactions, we would like to guarantee that there are few of them (about half). In order to guarantee that there are about as many red transactions as blue ones, we can add a (fictitious) item A_0 that is contained in all transactions. In that case, $|S_r| - |S_b|$ is also $O(\sqrt{n \log(2m)})$, and this implies that $|S_r| = n/2 + O(\sqrt{n \log(2m)}) = \frac{n}{2}(1 + O(\sqrt{\log(2m)/n}))$. Since $f(A_i; S_r) = r_i/|S_r| = 2r_i/n + O(\sqrt{\log(2m)/n})$, and $f(A_i; S) = (r_i + b_i)/n$, from the above bounds on $|r_i - b_i|$ we get that for each i,

$$|f(A_i; S_r) - f(A_i; S)| \leq \varepsilon(n, m) = O(\sqrt{\log(2m)/n}). \tag{6}$$

In practice, the halving method will work with any choice of δ_i, but the bounds on $|f(A_i; S_r) - f(A_i; S)|$ will not necessarily be guaranteed. In our implementation, we have found that setting

$$\delta_i = \sqrt{1 - \exp\Big(-\frac{\ln(2m)}{n}\Big)}$$

is very effective. The advantage is that if the defining parameters of the database, i.e., the number n of transactions and number m of items, are already known then the halving method requires a single scan of the database. It also guarantees that $|r_i - b_i|$ is at most $O(\sqrt{n \log(2m)})$ for any item, which is not as strong as $O(\sqrt{|S_i| \log(2m)})$, but still strong enough to guarantee the upper bound of (6).

2.2 The EASE algorithm

Implemented naively, t repeated rounds of halving would require t passes over the database, reducing the sample S to a sample S_0 of size approximately $\big||S|/2^t\big|$. The number of passes is governed by the final ε-approximation desired: the errors at every stage accumulate and hence the final error is bounded by $O(\varepsilon(n,m) + \varepsilon(n/2,m) + \cdots + \varepsilon(n/2^t,m)) = O(\varepsilon(n/2^t,m))$ since this is a geometric series. The multi-pass EA algorithm terminates when it detects that further subdivision will cause the distance to exceed the upper bound ε. This algorithm was described in [5] as multi-pass EA, and was the one implemented there.

Observe, however, that the halving process is inherently sequential in deciding the color of a transaction. In a single pass, we may store all the penalties of each halving method and proceed for each transaction as follows: based on the penalties of the first halving method, we decide whether to color that transaction red or not in the first sample. Should this transaction be red, we again compute the penalties of the second halving method, etc. until either the transaction is colored blue in a sample, or it belongs to the sample S_t. Thus all the repeated halving methods can be run simultaneously, in a single pass.

In Figure 2 we give the complete algorithm for the one-pass EASE algorithm. The algorithm is similar to multi-pass EA given in [5]. Some major differences are that the loop for halving ($k = 1...h$, where the user inputs the number h of halving times) is inside the loop for transactions (j). The user input h can never be more than $\lceil \log n \rceil$. Another difference is that for each halving stage an approximation is done regarding its δ. For $k = 1...h$ the initial $\delta_{i,k}$ for each transaction i is set to $\sqrt{1 - \exp(-2^{k-1}\ln(2m)/n)}$ assuming that after each halving the number of transactions is exactly halved. The loss in accuracy due to this approximation (as compared to the multiple-pass EA) is insignificant. This is confirmed by the experimental results shown later in this paper. The total time is reduced to some extent, but the major gain is due to the one-pass nature of this modification of multi-pass EA. Since EASE makes a final decision for each transaction, in effect, that transaction will never be looked at again, unless it is in the output (and we expect few of those). Since we need to keep the values of $Q_{i,1_k}$ and $Q_{i,2_k}$ (although the δ is independent of i in our choice of implementation), the memory complexity of EASE is only $O(mh) = O(m \log n)$, whereas that of the multi-pass EA is $O(|S|T_{max})$ which is the same as that of FAST. When m is bounded, T_{max} is generally bounded (on average for transactional databases, in the worst case for certain data streams), and $n \gg m$ can grow unbounded over time.

Moreover, a transaction needs to be processed on the fly in a given amount of time before the next transaction arrives, and this is the case with EA, as the time to process a transaction is bounded by $O(hT_{max})$ in the worst case. One important observation is that, on average, a transaction will go through *two* rounds of halving, hence the average cost of processing a transaction is only twice the average length of that transaction. Thus the runtime cost of EASE is linear in the size of the database (or its sample S in our case). This has important implications for data stream processing as well.

EASE(D, n, m, h)
obtain a simple random sample S from D;
for $k = 1$ to h
  $\delta_{i,k} = \sqrt{1 - \exp(-\frac{\ln(2m)}{\frac{n}{2^{k-1}}})}$;
  $Q_{i,1_k} = Q_{i,2_k} = 1$;
for each transaction j in S\{
  color transaction j red;
  for each halving k \{
    $Q^{(r)} = 0$; $Q^{(b)} = 0$;
    for each item i contained in j
      $Q^{(r)}_{i,1} = (1 + \delta_i)Q_{i,1_k}$;
      $Q^{(r)}_{i,2} = (1 - \delta_i)Q_{i,2_k}$;
      $Q^{(b)}_{i,1} = (1 - \delta_i)Q_{i,1_k}$;
      $Q^{(b)}_{i,2} = (1 + \delta_i)Q_{i,2_k}$;
      $Q^{(r)} += Q^{(r)}_{i,1} + Q^{(r)}_{i,2}$;
      $Q^{(b)} += Q^{(b)}_{i,1} + Q^{(b)}_{i,2}$;
    if $Q^{(r)} < Q^{(b)}$ then
      $Q_{i,1_k} = Q^{(r)}_{i,1}$;
      $Q_{i,2_k} = Q^{(r)}_{i,2}$;
    else
      color transaction j blue;
      $Q_{i,1_k} = Q^{(b)}_{i,1}$;
      $Q_{i,2_k} = Q^{(b)}_{i,2}$;
      exit for loop on k;
  \}
  if transaction j is red then
    set $S_0 = S_0 - \{j\}$
\}
return S_0;

Figure 2: The complete EASE algorithm.

3. FAST

Like EASE, the FAST algorithm [7] starts with an initial random sample S of D, and selects a subsample S_0 of S in which the high-level normalized aggregates are as close as possible to the high-level normalized aggregates in the database. When FAST was first applied to association-rule mining, these normalized aggregates were chosen to be 1-itemset frequencies. However, we shall show in Section 7, the aggregates can be chosen to represent 2-itemset frequencies or other categorical count statistics. Two approaches for computing the final small sample S_0 from S are given in [7]. They are, respectively, trimming and growing.

In this section, we discuss only the trimming method, which removes the "outlier" transactions from the sample S to obtain S_0. In this context an outlier is defined as a transaction whose removal from the sample maximally reduces (or minimally increases) the difference 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. Computing the supports of every item over S takes time at most $O(T_{\max} \cdot |S|)$, where $T_{\max}$ denotes the maximal transaction length. Because the cost of this step is relatively low, the sample S can be relatively large, thereby helping to ensure that the estimated supports are accurate.

To make the notion of difference between 1-itemset supports precise we define a distance function, based on the symmetric set difference, by setting

$$Dist_1(S_0, S) = \frac{|L_1(S) - L_1(S_0)| + |L_1(S_0) - L_1(S)|}{|L_1(S_0)| + |L_1(S)|} \quad (7)$$

FAST-TRIM$(D, k, Dist)$

obtain a simple random sample S from D;
compute $f(A;S)$ for each item A in 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 item A in S_0;
    set $S_0 = S_0 - \{t^*\}$, where $Dist(S_0 - \{t^*\}, S) = \min_{t \in G} Dist(S_0 - \{t\}, S)$;
  }
}
return S_0;

Figure 3: The FAST-TRIM Algorithm

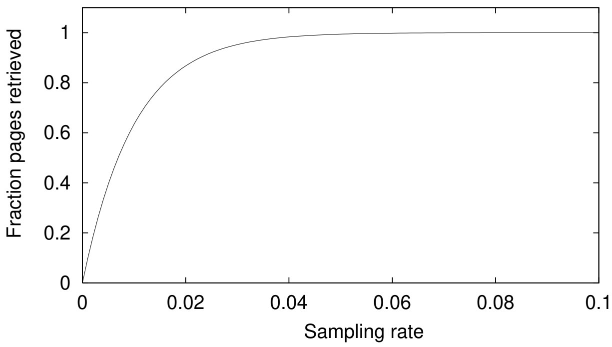

Figure 4: Fraction of pages fetched in transaction-level Bernoulli sampling.

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 $Dist_1$ takes values in $[0, 1]$, and that it 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. Other definitions of distance are possible. One is based on the L_2 metric between the frequency vectors:

$$Dist_2(S_0, S) = \sum_{A \in \mathcal{I}_1(S)} \big(f(A;S_0) - f(A;S)\big)^2. \qquad (8)$$

We have also considered the definition of distance based on the $Dist_\infty$ metric introduced in the previous section.

The basic FAST-TRIM algorithm is given in Figure 3. By choosing a value of k between 1 and $|S|$, the user can strike a balance between ineffective but cheap "oblivious" trimming and very effective but very expensive "pure greedy" trimming. For more details on different aspects of FAST such as variants of FAST, stopping criteria, detailed algorithm, see [7]. Efficient ways to estimate the distance functions and a complexity analysis of the trimming step in given in [5].

4. PRACTICAL SAMPLING CONSIDERATIONS

For EASE (or FAST) to be effective, they must be able to efficiently obtain an initial sample of a specified size. Assuming that the data is stored either in flat files or relational tables, the potentially high I/O costs of sampling pose the greatest challenge to algorithm performance.

For example, suppose we use a Bernoulli sampling scheme with rate q, in which each transaction is included into the sample with probability q and excluded with probability $1 - q$, independently of the other transactions. Many commercial database system support this type of sampling, as does the proposed ISO standard for sampling in SQL queries. Note that for a database size of N transactions, the sample size is random, having a binomial distribution with expected value Nq. To handle the fact that we want a fixed size sample of, say, k transactions, we can use a sampling rate $q > k/N$ such that with high probability the sample contains at least k transactions. We then sort the transactions in random order and use the first k of them. For example, the following SQL statement performs such a sampling operation, selecting 75 random transactions from a table containing 10,000 transactions:

```
WITH dt AS (
  SELECT *
  FROM transactions TABLESAMPLE BERNOULLI(1)
)
SELECT * FROM dt ORDER BY RAND()
FETCH FIRST 75 ROWS ONLY
```

In the query, the argument to `BERNOULLI` is the sampling rate expressed as a percentage and the `RAND` function returns a pseudo-random number between 0 and 1. In general, suppose that transactions are stored on disk in pages of size m and fetched into main memory a page at a time. Also suppose that the implementation of Bernoulli sampling is "ideal" in that no page is fetched more than once — this goal can be hard to achieve in current systems. Then the fraction f of pages fetched for a sampling rate of q is given by $f = 1 - (1 - q)^m$. Figure 4 displays f as a function of q for $m = 100$ rows per page. As can be seen, essentially all of the pages must be fetched whenever the sampling rate is higher than about 2%, incurring almost the same I/O cost as a full scan.

There are two approaches that can be used in the foregoing setting to sample efficiently. One approach is to simply store transactions in random order on disk, so that samples can be obtain as fast sequential disk scans. In many database systems, this can be done by "clustering" the data according to a random index. To keep the samples from becoming "stale," the data must periodically be reorganized, perhaps using an online REORG utility such as those found in a number of commercial database systems. In the database setting, the downside of this approach is the potential performance degradation for other queries posed to the system.

Another approach is to use a Bernoulli sampling scheme in which an entire page of transactions is included with probability q and excluded with probability $1-q$, independently of the other pages. A number of commercial systems support this sampling method, and a fixed-size sample of pages can be obtained using the over-sampling trick described above. The ratio of the I/O cost for a page-level Bernoulli sampling scheme with sampling rate q to the cost of a full scan is approximately equal to q. In our setting, this approach will work provided that the transactions are *unclustered* on disk in the sense that there is no statistical dependence between the data values in a transaction and the page in which the transaction resides. One way to test for clustering is to use the well-known Kiefer-Kolmogorov-Smirnov (KKS) non-parametric test for homogeneity of populations. The idea is as follows, assuming that each transaction is an l-vector of 0's and 1's. First take a random sample of k pages ($5 \le k \le 10$), denoted $B_1, B_2, \ldots, B_k$. Map each sampled transaction $t = (t_1, t_2, \ldots, t_l)$ to a real number, e.g., by setting $x_t = \sum_{i=1}^{l} 2^{i-1} t_i / 2^l$, and compute the statistic

$$U = \max_{t \in B} \sqrt{\sum_{i=1}^{k} n_i \big(F_i(x_t) - F(x_t)\big)^2},$$

where $B = B_1 \cup \cdots \cup B_k$, n_i is the number of transactions in B_i, $F_i(x)$ is the fraction of transactions $t \in B_i$ such that $x_t \le x$, and $F(x)$ is the fraction of transactions $t \in B$ such that $x_t \le x$. The statistic U takes on nonnegative values and is close to 0 if the data is unclustered; the larger the value of U, the less likely that the data is unclustered. More specifically, denote by $G(x; k)$ the probability that the KKS statistic exceeds x assuming that the data is unclustered. Assuming that the number of transactions in each page is larger than 20 or so, it can be shown that

$$G(x;k) \approx 1 - \frac{4}{\Gamma\big(\frac{k-1}{2}\big) 2^{(k-1)/2} x^{k-1}} \times \sum_{n=1}^{\infty} \frac{\gamma_n^{k-3} \exp[-\gamma_n^2 / 2x^2]}{J^2_{(k-1)/2}(\gamma_n)},$$

where Γ is the standard gamma function, J_ν is the Bessel function of the first kind of order ν, and $\gamma_1, \gamma_2, \ldots$ are the positive zeros of $J_{(k-3)/2}$. Thus if $G(U)$ is less than 0.2 or so, then the data can safely be assumed to be unclustered. The function G can be computed using standard numerical methods. An SQL query for computing $G(U)$ is given in Figure 5. For ease of exposition, the query is given for a table t that has a unique key column, and we assume that the 0-1 data are in columns c1–c3. In the query, `P_BERNOULLI` denotes page-level Bernoulli sampling,[2] `map` is a user-defined function (UDF) for converting the array of 0's and 1's to a real number, `KKSProb` is a UDF for evaluating the function G, and the `SAMPLE UNIT FOR` keywords specify the page number on which a sampled transaction resides.[3]

5. SHORT STUDY: APPROXIMATING FREQUENCIES

In this section, we illustrate the quality of the EASE subsamples compared to simple random sampling (denoted SRS in the figures), in the context of approximating the frequencies. We only report the results for EASE and SRS, not for FAST, but the results for FAST are very similar to EASE although not as good.

Computing frequencies of items can be done exactly by computing a histogram online, and even the frequencies of higher-order itemsets (e.g. for association-rule mining, see next section) can be computed online with excellent accuracy [10]. Note that approximating frequencies by a sample has its inherent limitations: it is highly unlikely that the estimated frequencies are exactly equal to the actual frequencies over the entire database. Moreover, EASE only guarantees a good approximation for the items that have been directly optimized by the halving method. This does not include higher-order itemsets, although we show in the next two sections that it does so in practice (with no guarantees).

[2]This keyword is nonstandard.

[3]The `SAMPLE UNIT FOR` keywords have been proposed by several vendors, but are not yet part of the official ISO proposal.

```
WITH
  dt1 AS (SELECT map(t.c1,t.c2,t.c3) AS xt, t.key as rid,
           SAMPLE UNIT FOR t AS pg
          FROM t TABLESAMPLE P\_BERNOULLI(1)),
  dt2 AS (SELECT pg AS i, COUNT(*) AS ni FROM dt1 GROUP BY pg),
  dt3 AS (SELECT COUNT(*) AS dv_cnt FROM dt2),
  dt4 AS (SELECT xt, pg, rid, i FROM dt1,dt2),
  dt5 AS (SELECT rid, xt, pg, i,
           (SELECT count(*) FROM dt4 AS s WHERE s.pg = dt4.i
            AND s.xt <= dt4.xt) / dv_cnt AS Fi
          FROM dt4,dt3),
  dt6 AS (SELECT rid,
           (SELECT count(*) FROM dt1 AS s WHERE s.xt <= dt1.xt)
           AS F FROM dt1),
  dt7 AS (SELECT COUNT(*) AS row_cnt FROM dt1),
  dt8 AS (SELECT dt5.rid as rid,
           SUM(dt2.ni*POWER((REAL(dt5.Fi)/dt2.ni)
             - (REAL(dt6.F)/dt7.row_cnt),2e0)) AS term
          FROM dt2,dt5,dt6,dt7
          WHERE dt5.rid = dt6.rid AND dt5.i = dt2.i
          GROUP BY dt5.rid),
  dt9 AS (SELECT MAX(term) AS maxterm FROM dt8)
SELECT SQRT(maxterm) AS kks, SMALLINT(dv_cnt) AS num_pages,
          KKSProb(SQRT(maxterm),SMALLINT(dv_cnt)) AS prob
   FROM dt9,dt3
```

Figure 5: An SQL query for the KKS test

The synthetic database was generated using code from the IBM QUEST project [2]. The parameter settings for synthetic data generation are similar to those in [2]: the total number of items was set to 1000, the number of transactions was set to 100,000, the number of maximal potentially frequent itemsets was set to 2000, the average length of transactions was 10, and the average length of maximal potentially frequent itemsets was 4. Following their conventions, the name of the data set is *T10.I4.100K*. This dataset will also be used in the next section.

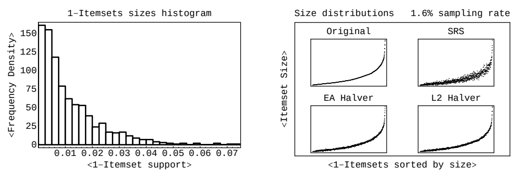

Figure 6: Items distribution, sorted by size, on original dataset (left) and on various samples (right).

In order to illustrate the quality of the EASE subsamples, we evaluate the absolute and relative accuracy of our samples, defined for each item i as

$$\begin{aligned} \mathrm{Abs}_i &= \Big| f(A; S_0) - f(A; S) \Big|, \\ \mathrm{Rel}_i &= \Big| 1 - f(A; S)/f(A; S_0) \Big|. \end{aligned}$$

Moreover, we also compute the root-mean-square (RMS) absolute and relative errors on all the items, defined as $\sqrt{\sum \mathrm{Abs}_i^2}$ and $\sqrt{\sum \mathrm{Rel}_i^2}$. For the methods tested, we evaluate two variants of EASE: the one described here and another (called L2 variant) where the penalty functions are simply

$$Q_i = \left(\frac{r_i}{r} - \frac{b_i}{b}\right)^2.$$

Although we cannot prove anything for L2, our results show that it is competitive and leads to slightly smaller RMS errors.

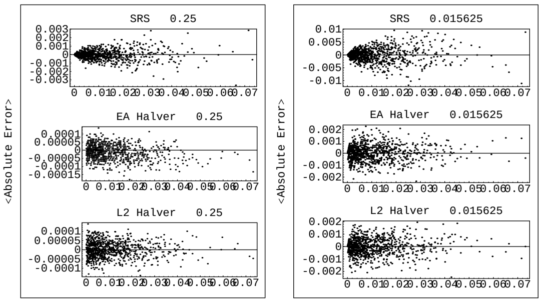

Figure 7: Absolute errors, sorted by items size.

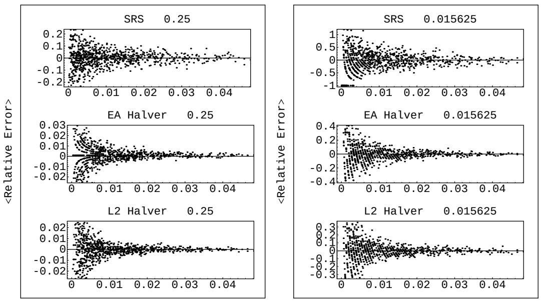

Figure 8: Relative errors, sorted by items size.

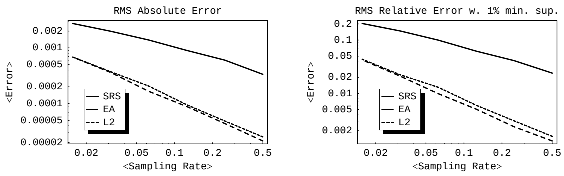

Figure 9: RMS error over all items, absolute (left) and relative (right).

Results. In Figure 6, we show the distribution of items, sorted by sizes, both in the original dataset and on the various samples (with a sampling ratio of 1.6%). In Figures 7 and 8, we plot the absolute and relative errors of the various samples, both for a sampling ratio of 25% (left) and for a ratio of 1.6% (right). *Please note the change in scale for* SRS *versus* EASE *variants.* Finally, in Figure 9 we plot the RMS absolute and relative errors. Overall, the figures show that EASE has an overall performance much better that simple random sampling.

6. CASE STUDY: MINING ASSOCIATION RULES

In this section we present an experimental comparison between FAST, multi-pass EA and EASE in the context of association-rule mining. In addition, we also performed experiments with simple random sampling (denoted SRS in the figures) in order to relate the current results to those previously reported in [7]. Since association rule algorithms [1, 9, 2] are focused on finding the frequent itemsets, i.e., the itemsets having minimum support, we have have used a performance metric similar to $Dist_1$ in order to measure the accuracy of our data reduction methods. In particular, accuracy is defined as :

$$accuracy = 1 - \frac{|L(D) - L(S)| + |L(S) - L(D)|}{|L(S)| + |L(D)|} \quad (9)$$

where, as before, $L(D)$ and $L(S)$ denote the frequent itemsets from the database D and the sample S, respectively. Notice that that accuracy is based on the set difference between all frequent itemsets generated from D and S, while $Dist_1$ is based only on frequent 1-itemsets. Early experimental evaluation by Zaki, et al. [14] had shown that random sampling was a suitable strategy for mining association rules. The results reported in [7] show that FAST achieves the same or higher accuracy (between 90% and 95%) using a final sample that is only a small fraction (15% to 35%) of a simple random sample.

To compare FAST and EASE, we used both synthetic and real-world databases in our experiments; we restrict ourselves here to reporting the results for the synthetic database and the trimming version of FAST. The real-world results will be reported in the full version of the paper. The synthetic database was generated using code from the IBM QUEST project [2], is labeled *T10.I4.100K* in their denomination, and is the same used in the previous section. We used a minimum support value of 0.77%, at which level there are a reasonable number of frequent itemsets and the length of the maximal frequent itemset is 6.

In order to make a fair comparison between the four algorithms we used Apriori in all cases to compute the frequent itemsets and used a common set of functions for performing I/O. We used a publicly available implementation of Apriori written by Christian Borgelt.[4] This implementation, which uses prefix trees, is reasonably fast and has been incorporated into a commercial data mining package. FAST was implemented using distance functions $Dist_1$ and $Dist_2$. For the parameter k, the group size in the FAST-TRIM algorithm, we chose a value of 10 since this was shown to be a reasonable choice in [7]. Both EASE and FAST start from a 30% simple random sample S, i.e., for FAST the trimming starts from S and for multi-pass EA and EASE the halvings start from S. As EASE cannot achieve all the sample sizes (because the halving process has a certain granularity), in each iteration we first ran EASE with a given ε value, and then used the obtained sample size to run FAST and SRS. EASE is not independent of the input sequence, so to account for any difference due to the particular input sequence the results of EASE are computed as an average over 50 runs, each one corresponding to a different shuffle of the input. In order to estimate the expected behavior of FAST and SRS, the

[4] `http://fuzzy.cs.uni-magdeburg.de/~borgelt/software.html`

results of these algorithms are also averaged over 50 runs, each time choosing a different simple random sample from the database.

Our primary metrics used for the evaluation are accuracy and execution time. The execution time is the total time that includes the time required for I/O, and that for finding the final sample and running Apriori.

Results. All the experiments were performed on a SUN Sparc Ultra workstation with a 333 MHz processor and 256MB memory. The final sampling ratios output by EASE were 0.93%, 1.85%, 3.7%, 7.5%, and 15.0% where the initial sample was 30% of the original data. Figure 10 (a) displays the accuracy of FAST-TRIM, one-pass and multi-pass EASE and SRS on the synthetic database as a function of the sampling ratio and Figure 10 (b) depicts the execution time of the above mentioned algorithms.

From Figure 10 (a) we observe that EASE and multi-pass EA achieve very good accuracy even for very small sample sizes. For Example, for a sample size of 1.85% they achieve more than 91% accuracy, while FAST with distance functions $Dist_1$ and $Dist_2$ achieves only 82% and 89.3% accuracies respectively. Similarly, for a final sample of 7.5% EASE achieves the maximum accuracy of 96.4% accuracy compared to 95.7% for multi-pass EA, 95.2% for FAST with $Dist_2$, 91.9% for FAST with $Dist_1$. SRS gives the worst accuracy for all sample sizes — for a sample of size 1.85% its accuracy is only 77%. Figure 10 (b) shows that except for SRS, EASE requires the least execution time among all methods. For example, for the above sample size of 1.85%, EASE requires 27% and 43% less execution time than FAST with distance functions $Dist_1$ and $Dist_2$ respectively, and 12% less than multi-pass EA. So, in summary, for mining association rules EASE is the superior data reduction method. It achieves better accuracy and requires smaller execution times compared to multi-pass EA and FAST. Furthermore, EASE can process transactions on the fly and has the smallest memory requirements of all the discussed methods.

We have not reported execution results for FAST-TRIM with $Dist_\infty$, because use of this distance function instead of $Dist_1$ or $Dist_2$ causes FAST-TRIM to run 8 times slower, which is consistent with the analysis of trimming given in [5].

We observe here that Toivonen's sampling-based association rule algorithm [12] requires a complete database scan. Toivonen's algorithm uses each scanned transaction to update a large number of count statistics and in addition requires an expensive step to eliminate false itemsets. In [7] we have shown that Toivonen's method is very accurate, but 10 times slower than FAST.

7. CASE STUDY: TESTING FOR INDEPENDENCE

As discussed in the introduction, our data reduction methods can be used to scale classical statistical procedures for count data to massive data sets. In this section we present some preliminary performance results when EASE is used to create a reduced data set that serves as input to a χ^2 two-way test for independence.

Consider a hypothetical set of financial survey data collected from 382,000 homeowners. We record for each homeowner whether the homeowner lives in an urban, suburban, or rural area, and whether or not the homeowner has refinanced their home within the past year. The survey results can be conceptually represented by a contingency table of counts as in Figure 11. The goal is to determine whether there is a statistical relationship between location and refinancing; such information would be useful to a mortgage lender when designing an advertising campaign. The "maximum likelihood" χ^2 test for independence computes the statistic

	Refinanced? Yes	No	Totals
Urban	120,000	80,000	200,000
Suburban	45,000	95,000	140,000
Suburban	30,000	12,000	42,000
Totals	195,000	187,000	382,000

Figure 11: Contingency table for survey data

$$\chi^2 = 2c_{\cdot\cdot}\log(c_{\cdot\cdot}) - \sum_{i=1}^{3} 2c_{i\cdot}\log(c_{i\cdot}) - \sum_{j=1}^{2} 2c_{\cdot j}\log(c_{\cdot j}) + \sum_{i=1}^{3}\sum_{j=1}^{2} 2c_{ij}\log(c_{ij}),$$

where c_{ij} is the count for region i and refinance response j, $c_{i\cdot}$ is the marginal count for region i, $c_{\cdot j}$ is the marginal count for refinance response j, and $c_{\cdot\cdot}$ is the total count for the entire table ($c_{\cdot\cdot}$ = 382,000). If χ^2 is sufficiently "large" — as determined by the performance parameters of the test together with statistical tables — then the "null hypothesis" of independence is rejected, and a statistical relationship is asserted. For our particular example, we have $\chi^2 = 34114$; for any reasonable choice of performance parameters, a χ^2 value of 34114 implies that there is a statistical relationship between location and refinancing.

For ease of exposition, suppose that the survey response for each homeowner is stored as a transaction consisting of a 5-vector of 0's and 1's. Each transaction has exactly two components equal to 1. A value of 1 in positions 1, 2, or 3 corresponds to an urban, suburban, or rural location, respectively, and a value of 1 in positions 4 or 5 corresponds to a refinance or no refinance, respectively, during the past year.

To make the χ^2 test scalable to large numbers of transactions, we can run the test on a sample of the data. The idea is to take a $100p$% sample and estimate the entries of the contingency table for the entire data set as $\hat{c}_{ij} = c^*_{ij}/p$, $\hat{c}_{i\cdot} = c^*_{i\cdot}/p$, and so forth, where c^* denotes a count computed from the reduced data set. Denote the test statistic computed from this estimated contingency table as X^2. The quantity X^2 is an asymptotically unbiased estimator of the true value χ^2.

Results. Figure 12 compares the error of X^2 as an estimator of χ^2 when X^2 is computed from samples created by EASE, FAST, and SRS. For our experiments, we use versions of EASE and FAST that try to make the estimates of the normalized counts $c_{ij}/c_{\cdot\cdot}$ for $i, j = 1, 2$ based on S_0 close to the corresponding estimates based on S. These normalized counts correspond to 2-itemset supports in the setting of association-rule mining. In the case of EASE, we run the

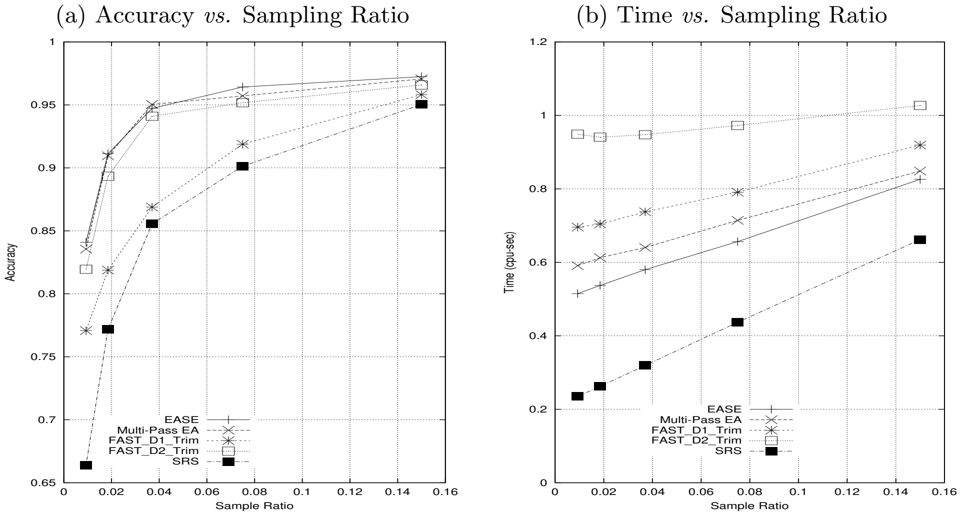

Figure 10: Results for synthetic dataset

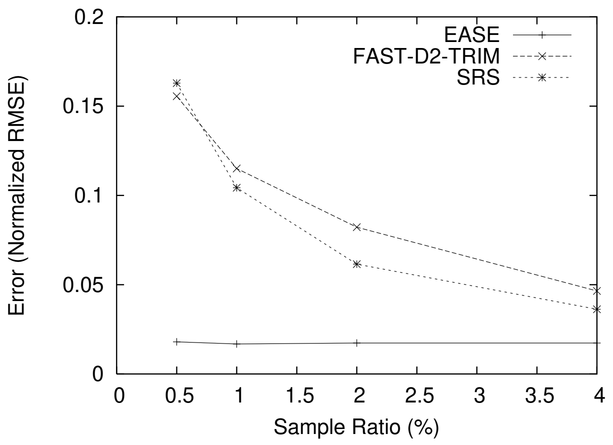

Figure 12: Results for χ^2 test

algorithm as usual, but each A_i now corresponds to a "2-itemset" rather than an individual item. In FAST, we use the distance function

$$Dist'_2(S_0, S) = \max_{A \in \mathcal{I}_2(S)} \Big| f(A; S_0) - f(A; S) \Big|,$$

when trimming. Our error metric is the *root mean square* (RMS) error, defined as the square root of the expected value of the quantity $(\chi^2 - X^2)^2$. In the figure, we normalize the RMS error by the true value of χ^2 for the entire database (i.e., by 34114). We estimate the expected value as the average over 50 independent sampling runs. Results are shown for several sampling ratios.

As can be seen from the figure, EASE dramatically outperforms both FAST and SRS for all sampling ratios studied. FAST and SRS perform comparably, with the errors of FAST somewhat higher than those of SRS except at the lowest sampling ratio.

8. CONCLUSION & FUTURE DIRECTIONS

We have presented a new data-reduction algorithm, EASE, that is tailored to categorical count data. The algorithm is similar to the previously-proposed FAST data-reduction algorithm in that the final sample is obtained by deterministic methods from a larger initial sample. EASE, however, uses an ε-approximation approach to obtain the final sample. We have empirically evaluated the performance of EASE in the context both of association-rule mining and classical contingency-table analysis. In all of our experiments, EASE outperforms both FAST and SRS, both in terms of speed and accuracy. We plan to perform more exhaustive experiments involving additional data sets and other variants of both EASE and FAST.

We are currently considering a variety of possible extensions to EASE. One extension modifies the halving round to pick red elements with a probability proportional to the desired final sample size, eliminating the need for storing several levels of penalties. The resulting algorithm would process transactions in worst case time $O(T_{max})$ with a memory requirement of $O(m)$.

Another interesting extension involves obtaining reduced samples of a datacube in which the measure is not merely a count, but a sum, average, or perhaps a higher moment such as variance.

We also plan to explore a line of research that focuses on cross-fertilizing EASE with ideas from Lossy Counting [10]. In particular, although EASE currently maintains penalties for all m items, it actually suffices to maintain penalties only for the "heavy" items (those with support greater than ε). A very crude approach would consist of running both EASE and Lossy Counting in parallel, and adding new penalties each time a heavy item is detected by Lossy Counting. This approach would bring the memory requirements of EASE in line with those of Lossy Counting. Interesting questions then arise with respect to the theoretical error guarantees of EASE.

Finally, we are considering some potential extensions of the EASE algorithm to permit maintenance of a sample of streaming data, and are currently investigating the details of such extensions.

Acknowledgments

Thanks to Alex Astashyn, the first author's student, who produced the graphs of section 5 as part of his Master's Thesis, and suggested the L2 variant of EASE.

9. ADDITIONAL AUTHORS

10. REFERENCES

[1] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, 1993.

[2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proceedings of International Conference on Very Large Databases (VLDB)*, 1994.

[3] N. Alon and J. H. Spencer. *The probabilistic method*. Wiley Interscience, New York, 1992.

[4] H. Bronnimann, B. Chazelle, and J. Matousek. Product range spaces, sensitive sampling, and derandomization. *SIAM Journal of Computing*, 1999.

[5] H. Bronnimann, B. Chen, M. Dash, and P. Haas, P. Scheuermann. Efficient data reduction method for on-line association rule discovery. In *Proceedings of NSF Workshop on New Generation of Data Mining*, 2002.

[6] B. Chazelle. *The discrepancy method*. Cambridge University Press, Cambridge, United Kingdom, 2000.

[7] B. Chen, P. Haas, and P. Scheuermann. A new two-phase sampling based algorithm for discovering association rules. In *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2002.

[8] J. Gray, A. Bosworth, A. Layman, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-total. In *Proceedings of 12th International Conference on Data Engineering (ICDE)*, pages 152–159, 1996.

[9] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, 2000.

[10] G. S. Manku and R. Motwani. Approximate frequency counts over data streams. In *Proceedings of International Conference on Very Large Databases (VLDB)*, 2002.

[11] J. Matousek. Derandomization in computational geometry. *Journal of Algorithms*, 20(3):545–580, 1996.

[12] H. Toivonen. Sampling large databases for association rules. In *Proceedings of International Conference on Very Large Databases (VLDB)*, 1996.

[13] L. G. Valiant. A theory of the learnable. *Communications of the Association for Computing Machinery*, 27(11):1134–1142, 1984.

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