

# Deterministic Algorithms For Sampling Count Data

Hüseyin Akcan, Alex Astashyn and Hervé Brönnimann

*Computer & Information Science Department*

*Polytechnic University, Brooklyn, NY 11201*

*hakcan01@cis.poly.edu, alex.astashyn@gmail.com and hbr@poly.edu*

---

## Abstract

Processing and extracting meaningful knowledge from count data is an important problem in data mining. The volume of data is increasing dramatically as the data is generated by day-to-day activities such as market basket data, web clickstream data or network data. Most mining and analysis algorithms require multiple passes over the data, which requires extreme amounts of time. One solution to save time would be to use samples, since sampling is a good surrogate for the data and the same sample can be used to answer many kinds of queries. In this paper, we propose two deterministic sampling algorithms, Biased-L2 and DRS. Both produce samples vastly superior to the previous deterministic and random algorithms, both in sample quality and accuracy. Our algorithms also improve on the run-time and memory footprint of the existing deterministic algorithms. The new algorithms can be used to sample from a relational database as well as data streams, with the ability to examine each transaction only once, and maintain the sample on-the-fly in a streaming fashion. We further show how to engineer one of our algorithms (DRS) to adapt and recover from changes to the underlying data distribution, or sample size. We evaluate our algorithms on three different synthetic datasets, as well as on real-world clickstream data, and demonstrate the improvements over previous art.

## 1 Introduction

Count data serve as the input for an important class of online analytical processing (OLAP) tasks, including association rule mining [1] and data cube online exploration [14]. These data are often stored in databases for further processing. However, the volume of data has become so huge that mining and analysis algorithms that require several passes over the data are becoming prohibitively expensive. Sometimes, it is not even feasible (or desirable) to store it in its entirety, e.g., with network traffic data. In that case, the data must be processed as a stream. For most OLAP tasks, exact counts are not required and an approximate representation is appropriate, motivating an approach called *data reduction* [4]. A similar trend was observed in traditional database management systems (DBMS) where exact results taking too long led to approximate query answering as an alternative [11, 16].

A general data reduction approach that scales well with the data is sampling. The data stream community also uses sampling as a representative for streaming data [3, 19]. Even though sampling is widely used for analyzing data, the use of random samples can lead to unsatisfactory results. For instance, samples may not accurately represent the entire data due to fluctuations in the random process. This difficulty is particularly apparent for small sample sizes and bypassing it requires further engineering.

The main product of this research consists of two deterministic algorithms, named below Biased-L2 and DRS, to find a sample  $S$  from a dataset  $D$  which optimizes

---

\* This work is partially supported by NSF CAREER Grant CCR-0133599. We thank Peter Haas, Peter Scheuermann, and Goce Trajcevski for their comments, as well as [20] for the BMS-Webview-1 dataset.

the root mean square (RMS) error of the frequency vector of items over the sample (when compared to the original frequency vector of items in  $D$ ). Both algorithms are a clear improvement over SRS (simple random sample) and other more specialized deterministic sampling algorithms such as FAST [8] and EASE [5]. The samples our algorithms produce can be used as surrogate for the original data, for various purposes such as query optimization, approximate query answering [11, 16], or further data mining (e.g., building decision trees or iceberg cubes). In this latter context, the items represent all the values of all the attributes in the DBMS and one wants to maintain, for each table, a sample which is representative for every attribute simultaneously. We assume here categorical attributes—numerical attributes can be discretized, e.g., by using histograms and creating a category for each bucket.

In Section 2 we talk about the previous work. Later, in Section 3 we present our sampling algorithms, Biased-L2 and Deterministic Reservoir Sampling (DRS), for deterministically sampling count data. In Section 4 we evaluate our algorithms (Biased-L2 and DRS) on several real-world and synthetic datasets, with various criteria and settings. Finally, in Section 5 we finish with the concluding remarks.

## **Our Contributions**

- In this paper, we present two novel deterministic sampling algorithms: Biased-L2 and DRS, to sample count data (tabular or streaming).
- Both of our algorithms generate samples with better accuracy and quality compared to the previous algorithms (EASE and SRS).
- Our algorithms improve on previous algorithms both in run-time and memory footprint.

- We perform extensive simulations with synthetic and real-world datasets under various settings, and demonstrate the superiority of our algorithms.

## 2 Related Work

The survey by Olken and Rotem [22] gives an overview of random sampling algorithms in databases. Sampling is discussed and compared against other data reduction methods in the NJ Data Reduction Report [4]. In addition to sampling, a huge literature is available on histograms [15] and wavelet decompositions as data reduction methods, and we do not attempt to survey it here. We note however that sampling provides a general-purpose reduction method which simultaneously applies to a wide range of applications. Moreover, the benefits of sampling vs. other data reduction methods are increased with multi-dimensional data: the larger the dimension, the more compact sampling becomes vs., e.g., multi-dimensional histograms or wavelet decompositions [4]. Also, sampling retains the relations and correlations between the dimensions, which may be lost by histograms or other reduction techniques. This latter point is important for data mining and analysis.

Zaki *et al.* [25] state that simple random sampling can reduce the I/O cost and computation time for association rule mining. Toivonen [23] propose a sampling algorithm that generates candidate itemsets using a large enough random sample, and verifies these itemsets with another full database scan. Instead of a static sample, John and Langley [18] use a dynamic sample, where the size is selected by how much the sample represents the data, based on the application. The FAST algorithm introduced by Chen *et al.* [8] creates a deterministic sample from a relatively large initial random sample by trimming or growing a sample according to a local optimization criterion. The EASE algorithm by Brönnimann *et al.* [5] again uses a

relatively large sample and creates a deterministic sample by performing consecutive halving rounds on the sample. EASE algorithm keeps penalty functions for each item per each separate halving round. Each transaction has to pass the test at each level in order to be added to the sample. The penalties change based on the accept or reject decision of the transaction, and the goal is to generate a sample having item supports as close as possible to those in the dataset. Multiple halving rounds per transaction and the penalty functions used for each round introduces additional complexity to EASE compared to Biased-L2. The Biased-L2 algorithm we present in this paper uses ideas similar to EASE based on discrepancy theory [7], but samples the dataset without introducing halving rounds and improves on the run-time and memory requirements, as well as the sample quality and accuracy. Although in this paper we focus on count data that occur mostly in database settings, Biased-L2 algorithm is generic for any discretized data, and in [2] it is applied to sampling geometric point data for range counting applications.

The main difference between DRS and FAST is that DRS keeps a smaller sample in memory, examines each transaction only once, and it is suitable to handle streaming data. DRS algorithm uses a cost function based on RMS distance which is incrementally updated by changes to the sample. In this paper we only give the incremental formulas specific to our case, where the sample size does not change by updates. Additional incremental formulas for various distance functions are presented in [6]. As the sample size can be preset exactly, DRS does not have accuracy problems caused by the halving rounds of EASE. Johnson *et al.* [19] suggest that, for cases when the stream size is unknown, it is useful to keep a fixed-sized sample. Since in practice most stream sizes are unknown, this can be best done by allowing the algorithms to dynamically remove transactions from the sample, as in reservoir sampling [24] and DRS.

Gibbons *et al.* [12] propose concise sampling and introduce algorithms to incrementally update a sample for any sequence of deletions and insertions. While concise sample dramatically reduces memory footprint, it works for single attribute sampling, lacking the ability to give any correlation between attributes, which is desirable for multi-dimensional data.

Vitter [24] introduces reservoir sampling, which allows random sampling of streaming data. Reservoir sampling produces a sample of quality identical to SRS, but does not examine all the data. Whenever a new record is selected, it evicts a random record. In contrast, DRS adapts to changes in distribution by deterministically selecting the worst record to evict. Gibbons *et al.* [13] use reservoir sampling as a backing sample to keep the histograms up to date under insertions and future deletions.

In [19] different approximation algorithms are discussed including Reservoir Sampling [24], Heavy Hitters algorithm [21], Min-Hash Computation [9] and Subset-Sum sampling [10]. Among these we only compare our algorithms with Reservoir Sampling (random sampling in general), since the rest of the algorithms are tailored for specific applications.

### **3 Deterministic Sampling Algorithms**

In this section we first describe the notation used throughout the paper, and then present our deterministic sampling algorithms: Biased-L2, and Deterministic Reservoir Sampling, respectively in Section 3.2, and Section 3.3

### 3.1 Notation

Let  $D$  denote the database of interest,  $d = |D|$  the number of transactions,  $S$  a deterministic sample drawn from  $D$ , and  $s = |S|$  its number of transactions. We denote by  $I$  the set of all items that appear in  $D$ , by  $m$  the total number of such items, and by  $\text{size}(j)$  the number of items appearing in a single transaction  $j \in D$ . We let  $T_{\text{avg}}$  denote the average number of items in a transaction, so that  $dT_{\text{avg}}$  denotes the total size of  $D$  (as counted by a complete item per transaction enumeration).

In the context of association rule mining, an *itemset* is a subset of  $I$ , and we denote by  $\mathcal{I}(D)$  the set of all 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  $j \in D$ . A  $k$ -itemset is an itemset with  $k$  items, and their collection is denoted by  $\mathcal{I}_k(D)$ ; in particular the 0-itemset is the empty set (contained in all the transactions) and the 1-itemsets are simply the original items. Thus  $\mathcal{I}(D) = \cup_{k \geq 0} \mathcal{I}_k(D)$ . The itemsets over a sample  $S \subseteq D$  are  $\mathcal{I}(S) \subseteq \mathcal{I}(D)$ , and  $\mathcal{I}_k(S)$  is defined similarly.

For a set  $T$  of transactions and an itemset  $A \subseteq I$ , we let  $n(A; T)$  be the number of transactions in  $T$  that contain  $A$  and  $|T|$  the total number of transactions in  $T$ . Then the support of  $A$  in  $T$  is given by  $f(A; T) = n(A; T)/|T|$ . In particular,  $f(A; D) = n(A; D)/|D|$  and  $f(A; S) = n(A; S)/|S|$ . Given a threshold  $t > 0$ , an item is frequent in  $D$  (resp. in  $S$ ) if its support in  $D$  (resp.  $S$ ) is no less than  $t$ .

The distance between two sets  $D$  and  $S$  with respect to the 1-itemset frequencies can be computed via the discrepancy of  $D$  and  $S$ , defined as

$$\text{Dist}_\infty(D, S) = \max_{A \in I} |f(A, D) - f(A, S)|. \quad (1)$$

A sample  $S$  such that  $\text{Dist}_\infty(D, S) \leq \varepsilon$  is called an  $\varepsilon$ -approximation. Other ways

to measure the distance of a sample are via the L1-norm or the L2-norm (also called ‘root-mean-square’ - RMS),

$$Dist_1(D, S) = \sum_{A \in I} |f(A, D) - f(A, S)|, \quad (2)$$

$$Dist_2(D, S) = \sqrt{\sum_{A \in I} (f(A, D) - f(A, S))^2}. \quad (3)$$

In order to measure the accuracy of the sample  $S$  for evaluating frequent itemset mining, as in [5, 8] the following measure is used:

$$Accuracy(S) = 1 - \frac{|L(D) \setminus L(S)| + |L(S) \setminus L(D)|}{|L(S)| + |L(D)|}, \quad (4)$$

where  $L(D)$  and  $L(S)$  represent the number of frequent itemsets in dataset and sample.  $L(D) \setminus L(S)$  represents the number of itemsets exist in dataset but not in sample, and  $L(S) \setminus L(D)$  the other way around.

### 3.2 Biased-L2

Biased-L2 algorithm examines each transaction in sequence, and builds up a sample with a fixed sampling rate of  $\alpha$ . Each transaction is examined only once, in accordance with the streaming model, and either kept in the sample (accepted) or dropped out of the sample (rejected). The decision to keep or drop a transaction is deterministic, and based on the combined approximation properties for every item. Namely, the algorithm maintains a penalty function per item  $i$  based on the number  $n_i$  of transactions (so far) containing that item and the corresponding number  $r_i$  for the selected sample. Each penalty function minimizes when the item frequency over the sample equals that over the data set, and increases sharply when the item is under- or over-sampled. The decision to keep or reject a transaction induces a

change in the penalties, and the transaction is kept if the total penalty is not increased, and dropped otherwise. The penalty function for each item  $i$  is defined as follows:

$$Q_i = (r_i - \alpha n_i)^2 - n_i \alpha (1 - \alpha).$$

The first term in the equation is the L2-distance between the dataset and the sample. The second term in the equation is used to ensure that there is a choice of accepting or rejecting a transaction such that the penalty is not increased (without it, the penalty would always increase when  $r_i = \alpha n_i$ , regardless of accepting or rejecting a transaction). The total penalty for a transaction  $j$  is  $Q = \sum_{i \in j} Q_i$ , and the decision whether or not to keep  $j$  is made by trying to minimize  $\Delta Q$ . When a transaction is accepted, both  $r_i$  and  $n_i$  are incremented but when it is rejected only  $n_i$  is incremented. Therefore, the  $\Delta Q$  function for a given item  $i$  becomes:

$$\Delta Q_i^{accept} = (\alpha - 1)(-1 + 2\alpha(n_i + 1) - 2r_i),$$

$$\Delta Q_i^{reject} = \alpha(-1 + 2\alpha(n_i + 1) - 2r_i).$$

By choosing transactions such that  $\Delta Q_i^{accept} \leq \Delta Q_i^{reject}$ , we accept a transaction if  $\sum_{i \in j} (-1 + 2\alpha(n_i + 1) - 2r_i) \leq 0$ .

The complete code of the Biased-L2 algorithm is presented in Figure 1. Since  $n_i$  is incremented regardless of whether the transaction is accepted or not, incrementing it ahead of time leads to the simplified acceptance condition given in line 9 of the pseudocode.

**Theorem 1.** *The Biased-L2 algorithm with sampling ratio  $\alpha$  produces a sample of discrepancy  $\varepsilon$  and size  $\alpha d(1 \pm \varepsilon)$ , where  $\varepsilon = O\left(\sqrt{(1 - \alpha)m/(\alpha d)}\right)$ . The running time is  $O(dT_{\text{avg}})$  and the space complexity is  $O(m + \alpha dT_{\text{avg}})$ .*

*Proof.* The overall penalty is zero at first, and never increases during the sampling

process. Thus,

$$\sum_{i \in I(D)} (r_i - \alpha n_i)^2 - n_i \alpha (1 - \alpha) \leq 0$$

Rearranging the terms,

$$\sum_{i \in I(D)} (r_i - \alpha n_i)^2 \leq \alpha (1 - \alpha) \sum_{i \in I(D)} n_i$$

On the right hand side, we have the count of every item. Every term on the left is non-negative, which implies that every term on the left is less than or equal to the whole right hand side. Hence,

$$r_i = \alpha n_i + O\left(\sqrt{\alpha(1-\alpha)dm}\right).$$

If we add a sentinel item to each transaction, and divide by  $\alpha d$ , we get the final supports as:

$$f(A, S) = f(A, D) + O\left(\sqrt{(1-\alpha)m/(\alpha d)}\right).$$

□

### Better Theoretical Bounds

Biased-L2 uses a quadratic cost function. However, we can improve on the theoretical discrepancy bound if we use an exponential cost function,  $Q_i = Q_{i,1} + Q_{i,2}$ , with:

$$Q_{i,1} = (1 + \delta)^{r_i} (1 - \delta)^{\alpha n_i},$$

$$Q_{i,2} = (1 - \delta)^{r_i} (1 + \delta)^{\alpha n_i}.$$

We call the algorithm using the new cost function as Biased-EA. Based on this new cost function, we can prove the following theorem:

**Theorem 2.** *The Biased-EA algorithm with sampling ratio  $\alpha$  produces a sample of discrepancy  $\varepsilon$  and size  $\alpha d(1 \pm \varepsilon)$ , where  $\varepsilon = O\left(\sqrt{\log(2m)/(\alpha d)}\right)$ . The running time is  $O(dT_{\text{avg}})$  and the space complexity  $O(m + \alpha dT_{\text{avg}})$ .*

BIASED-L2 ( $D, \alpha$ )

```

1:  $S_{Biased-L2} \leftarrow \emptyset$ 
2: for each item  $i$  in  $D$  do
3:    $n_i \leftarrow r_i \leftarrow 0$ 
4:   for each transaction  $j$  in  $D$  do
5:      $sum_r \leftarrow sum_n \leftarrow 0$ 
6:     for each item  $i$  in  $j$  do
7:        $n_i \leftarrow n_i + 1$ 
8:        $sum_r \leftarrow sum_r + r_i; sum_n \leftarrow sum_n + n_i$ 
9:       if  $size(j)/2 + sum_r - \alpha \cdot sum_n \leq 0$  then
10:        /* Keep the transaction */
11:        Insert  $j$  into  $S_{Biased-L2}$ 
12:        for each item  $i$  in  $j$  do
13:           $r_i \leftarrow r_i + 1$ 
14:   return  $S_{Biased-L2}$ 

```

Fig. 1. The Biased-L2 algorithm

*Proof.* In the beginning,  $r_i = 0, n_i = 0$  and  $Q_i = 2$ .  $Q_i^{(init)} = 2m$ , hence  $Q_i^{(final)} \leq 2m$  for each item  $i$ , since all terms are positive. Therefore,

$$(1 + \delta_i)^{r_i - \alpha n_i} (1 - \delta_i)^{\alpha n_i} (1 + \delta_i)^{\alpha n_i} + (1 - \delta_i)^{r_i - \alpha n_i} (1 + \delta_i)^{\alpha n_i} (1 - \delta_i)^{\alpha n_i} \leq 2m,$$

Rearranging the terms,

$$(1 + \delta_i)^{r_i - \alpha n_i} + (1 - \delta_i)^{r_i - \alpha n_i} \leq 2m / (1 - \delta_i^2)^{\alpha n_i}$$

Since both terms are positive,

$$(1 + \delta_i)^{r_i - \alpha n_i} \leq 2m / (1 - \delta_i^2)^{\alpha n_i},$$

$$(1 - \delta_i)^{r_i - \alpha n_i} \leq 2m / (1 - \delta_i^2)^{\alpha n_i}.$$

Taking logarithms and combining the inequalities, we get:

$$r_i = \alpha n_i \pm \frac{1}{\log(1 - \delta_i)} (\log(2m) - \alpha n_i \log(1 - \delta_i^2)) \quad (5)$$

We want to minimize the error in terms of  $\delta_i$ . Approximating  $\log(1 - \delta_i)$  with  $-\delta_i$  and  $\log(1 - \delta_i^2)$  with  $-\delta_i^2$ , and taking the derivative, we find that a good choice for  $\delta_i$  is

$$\delta_i = \sqrt{\log(2m)/(\alpha n_i)}.$$

Substituting in equation (5) yields

$$r_i = \alpha n_i \pm 2\sqrt{\log(2m)(\alpha n_i)} = \alpha n_i + O(\sqrt{\alpha n_i \log(2m)}).$$

If we add a sentinel item to each transaction, and divide by  $\alpha d$ , we get the final supports as:

$$f(A, S) = f(A, D) + O(\sqrt{\log(2m)/(\alpha d)}).$$

□

Even though theoretically Biased-EA gives better discrepancy *bounds* than Biased-L2, the somewhat complex cost function and the requirement to carefully select the input parameters make it less practical to implement, especially in streaming cases, where processing speed is important and data distribution is unknown. For these reasons, we only give Biased-EA for theoretical interest (also for special cases where  $m$  is quite large, e.g. geometric data [2]), and continue using Biased-L2 as our algorithm of choice throughout the paper, because of its simplicity.

In this section, we presented Biased-L2 as the first of our deterministic sampling algorithms. Biased-L2 works in the streaming model, where each transaction is examined only once and a sample with a given rate  $\alpha$  is created from the underlying stream. The algorithm is superior to EASE both in run-time and memory require-

ments, since EASE works on halvings where each transaction is examined  $O(\log d)$  times, and therefore  $O(m \log d)$  counters have to be kept.

### 3.3 Deterministic Reservoir Sampling (DRS)

In this section, we present Deterministic Reservoir Sampling algorithm. The main idea is to maintain a sample of constant size  $s$  and periodically add a transaction while evicting another. The choice of transactions is computed in order to keep a distance function as small as possible (here, we present the algorithm using  $Dist_2$ ). In particular, it differs from EASE and Biased-L2 by its ability to not only add new transactions to the sample but also remove undesired transactions. As we will show, this ability makes the sample more robust to changes of the distribution in the streaming or tabular data scenarios.

The algorithm maintains the *worst* transaction  $W$  in the sample, i.e., the one whose removal from  $S$  decreases  $Dist_2(D, S)$  the most. An *update* by  $T$  consists of replacing  $W$  by some transaction  $T$ . The parameter  $k$  is used to control the number of updates as follows: The algorithm scans the consecutive transactions in blocks of size  $k$  and for each block, computes the best transaction  $T$  for an update, i.e., such that  $Dist_2(D, (S \setminus \{W\}) \cup \{T\})$  is minimized. One important observation here is that even replacing  $W$  by the best  $T$  may not decrease the cost function in some situations. In this case, the sample is kept unchanged for this block.

The full algorithm is presented in Figure 2. This version of the algorithm works in a single pass and updates the supports of items on the fly, both in the dataset and in the sample. The only requirement for single pass is that the size of the dataset must be known in advance, in order to compute the  $Dist_2$  function. For the streaming

case, or other cases where we do not know the size of the dataset in advance, a slight modification to the algorithm is possible, that starts with an expected dataset size and zero frequencies, and gradually increases them during run-time.

Since at each update, only one transaction is added and another removed from the sample, limited number of items on average are affected from this change, allowing us to easily update the penalty function incrementally. The difference in penalty function after adding transaction  $T$  is:

$$\Delta_T = \frac{\text{size}(T)}{s^2} + \frac{2}{s} \sum_{i \in T} \left( f(A_i, S_{DRS}) - f(A_i, D) \right), \quad (6)$$

Similarly, the difference after removing transaction  $W$  is

$$\Delta_W = \frac{\text{size}(W)}{s^2} - \frac{2}{s} \sum_{j \in W} \left( f(A_j, S_{DRS}) - f(A_j, D) \right), \quad (7)$$

By adding these two differences together, we can find the total difference caused by the update since the sample size doesn't change:

$$\begin{aligned} \text{Dist}_2(D, S_{DRS} \cup \{T\} \setminus \{W\}) &= \frac{\text{size}(T) + \text{size}(W)}{s^2} \\ &+ \frac{2}{s} \sum_{i \in T \setminus W} \left( f(A_i, S_{DRS}) - f(A_i, D) \right) \\ &- \frac{2}{s} \sum_{j \in W \setminus T} \left( f(A_j, S_{DRS}) - f(A_j, D) \right). \quad (8) \end{aligned}$$

Based on Equation (8) the computation can be done in time  $O(T_{\text{avg}})$  per transaction. Thus the run-time cost of one iteration of the loop is  $O(T_{\text{avg}})$  except if it triggers an update, in which case it becomes  $O(sT_{\text{avg}})$  since each transaction in the sample needs to be re-examined to find the new worse transaction. In order to describe the overall running time, we consider the choice of  $k$ . A choice of  $k = 1$  means that we update in a totally greedy fashion (steepest descent), which might perform well in terms of error but might be very expensive in terms of run-time. A choice of

**DRS** ( $D, k$ )

- 1:  $S_{DRS} \leftarrow$  first transactions of  $D$
- 2:  $N \leftarrow 0; C \leftarrow Dist_2(D, S_{DRS})$
- 3:  $W \leftarrow \text{FINDWORSE}(D, S_{DRS})$
- 4:  $C_{min} \leftarrow \infty; T_{min} \leftarrow \emptyset$
- 5: **for** each transaction  $j$  in  $D$  **do**
- 6:    $N \leftarrow N + 1$
- 7:    $C_{new} = Dist_2(D, S_{DRS} \cup \{j\} \setminus \{W\})$
- 8:   **if**  $C_{new} < C_{min}$  **then**
- 9:      $C_{min} \leftarrow C_{new}; T_{min} \leftarrow j$
- 10:   **if**  $N \equiv 0 \pmod k$  **then**
- 11:     */\* periodical updates, after every  $k$  transactions \*/*
- 12:     **if**  $C_{min} < C$  **then**
- 13:        $S_{DRS} \leftarrow S_{DRS} \cup \{T_{min}\} \setminus \{W\}$
- 14:        $C \leftarrow C_{min}$
- 15:        $W \leftarrow \text{FINDWORSE}(D, S_{DRS})$
- 16:      $C_{min} \leftarrow \infty; T_{min} \leftarrow \emptyset$
- 17: **return**  $S_{DRS}$

**FINDWORSE** ( $D, S_{DRS}$ )

- 1:  $C_w \leftarrow \infty; W \leftarrow \emptyset$
- 2: **for** each transaction  $j$  in  $S_{DRS}$  **do**
- 3:   **if**  $Dist_2(D, S_{DRS} \setminus \{j\}) < C_w$  **then**
- 4:      $W \leftarrow j; C_w \leftarrow Dist_2(D, S_{DRS} \setminus \{j\})$
- 5: **return**  $W$

Fig. 2. The DRS algorithm

$k = d$  means no updates. In between these extremes, selecting a bigger value will decrease the number of updates on the sample and speed up the sampling process, but decrease the quality of the sample. Selecting a smaller value will slow down the process while increasing the quality of the sample. Ultimately, we should pick the smallest value of  $k$  which affords a reasonable running time. Following the analogy with reservoir sampling [24], we could hope that the number of updates is  $O(\log d/s)$ , yet our updates are dictated by a comparatively more complex process, and this hope is not borne out by the experiments. Instead, the actual bounds seems closer to the trivial upper bound of  $d/k$ . A good compromise seems  $k = s/c$  for some constant  $c > 0$ , which implies a total number of updates which is  $O(d/s)$  and thus a total run-time of  $O(dT_{\text{avg}})$ . Empirical results are given in the experiments section, showing the effect of  $k$  on the overall sample quality and accuracy.

As for memory requirements, DRS needs to store the frequency counts of every item separately in  $D$  and  $S_{DRS}$ , as well as the sample, hence has a space complexity of  $O(m + sT_{\text{avg}})$ , which is equivalent to that of Biased-L2 (since finally  $s = \alpha d$ ).

In Sections 3.2 and 3.3, we presented our deterministic sampling algorithms, Biased-L2 and Deterministic Reservoir Sampling. Among these two, Biased-L2 is our algorithm of choice if we need speed and simplicity, and DRS is preferred when we need exact sample sizes, or when the underlying dataset distribution changes and fast recovery is needed.

## 4 Experimental Results

In this section, we compare the new algorithms (Biased-L2, DRS) with the previous ones on various datasets, and show the superiority of the new algorithms in terms

Dataset	NoOfTrans	NoOfItems	AvgTransSize	Apr.Supp.	1-itemsets	2-itemsets	3-itemsets	4-itemsets
T5I3D100K	100000	1000	5	0.4	460	84	39	18
T10I6D100K	100000	1000	10	0.4	633	774	445	378
T50I10D100K	100000	1000	50	0.75	832	45352	40241	45140
BMS1	59602	497	2.5	0.3	225	169	39	0

Fig. 3. Dataset parameters

of sample quality and accuracy. We also highlight additional features of the DRS algorithm using tailored experiments.

#### 4.1 Datasets used

The datasets used in our experiments are three synthetic datasets from IBM [17] (T5I3D100K, T10I6D100K, T50I10D100K), and one real-world clickstream dataset (BMS-WebView-1 or BMS1) [20]. Both types of datasets are count datasets, with variable length transaction sizes. The detailed parameter information of each dataset is listed in Figure 3. The reason for our choice of datasets is to have different maximum lengths of a transaction and of an itemset, in order to evaluate the dependency on these parameters and make sure the results don't differ too much. BMS1 acts as the real-world/typical control data set. More detailed information about the BMS1 dataset can be found in [20].

#### 4.2 Sampling count data

In this section, we compare the results of the simple random sample (SRS), EASE, Biased-L2, and DRS algorithms on the association rule datasets. The comparison is based on the quality and accuracy of the sample, given the cost function in Eq. (3)

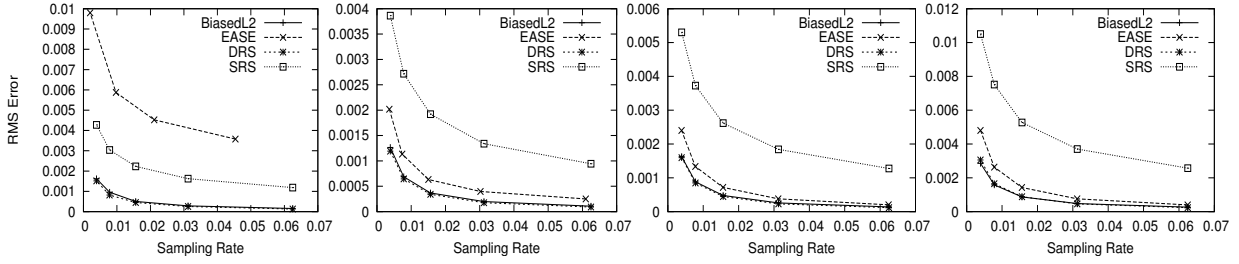


Fig. 4. RMS error of SRS, EASE, Biased-L2, and DRS for four datasets (BMS1, T5, T10, T50).

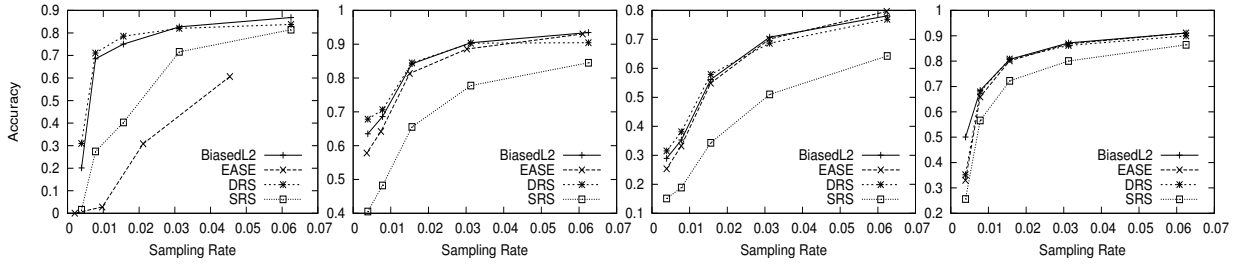


Fig. 5. Accuracies of SRS, EASE, Biased-L2, and DRS for four datasets (BMS1, T5, T10, T50).

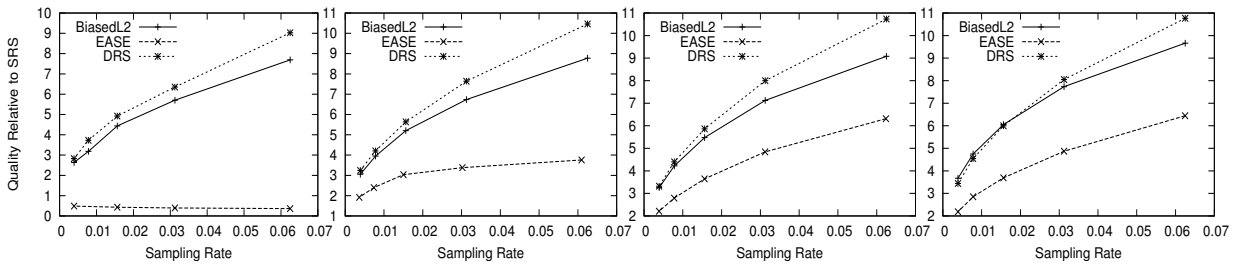


Fig. 6. Ratio of the RMS error of SRS over EASE, Biased-L2, and DRS for four datasets (BMS1, T5, T10, T50). Higher  $y$ -coordinate values correspond to better quality samples and the accuracy function in Eq. (4).

Figure 4 plots the RMS error, and Figure 5 the accuracy results of SRS, EASE, Biased-L2, and DRS on all four datasets. The algorithms are run with sampling rates of 0.003, 0.007, 0.015, 0.03, and 0.062, or the sample size equivalent of them, based on the size of the dataset. For synthetic datasets, the algorithms are run 50 times with a random shuffle of  $D$ , and the average is calculated. For the real-

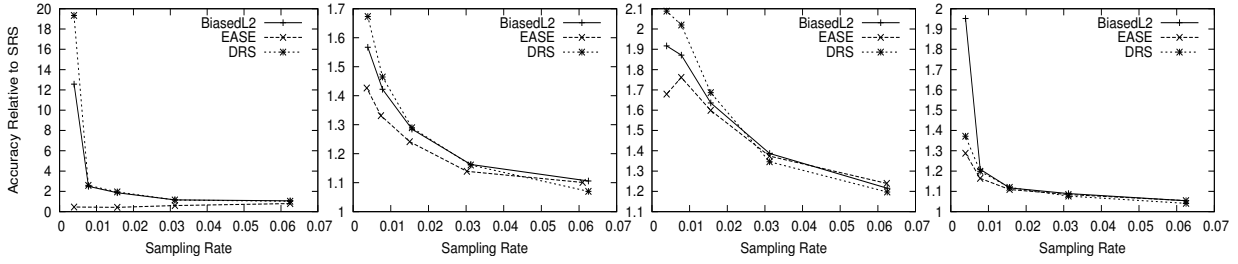


Fig. 7. Ratio of the accuracy of EASE, Biased-L2, and DRS over SRS for four datasets (BMS1, T5, T10, T50). Higher  $y$ -coordinate values correspond to more accurate samples world dataset (BMS1), the original order of transactions is kept, and deterministic algorithms (EASE, Biased-L2, and DRS) are run once, while random sampling algorithm (SRS) is again run 50 times.

Figure 6<sup>1</sup> presents the ratio comparison of results in Figure 4 relative to SRS for each dataset. From these results, we can say that the sample quality of DRS and Biased-L2 are superior compared to the results of EASE and SRS. In terms of RMS error, on average, DRS is a factor of 14 times and Biased-L2 is a factor of 12 times better than EASE on the real-world dataset, and a factor of 2 times better on synthetic datasets. On average, the new algorithms are also a factor of 6 times better than SRS on all datasets.

In order to compare the accuracy of the samples, we use the Apriori [1] algorithm to generate association rules both for the dataset and the samples. Later, we use Equation (4) to calculate accuracies. Figure 5 plots the accuracy results of SRS, EASE, Biased-L2 and DRS on all datasets. In addition, Figure 7<sup>1</sup> presents the average ratio comparison of the accuracy results for each dataset, based on the SRS accuracy for each sampling rate. From the figures we can say that on average, DRS

<sup>1</sup> EASE algorithm was unable to generate the expected sample sizes on BMS1 dataset, the accuracy and quality ratio values of EASE algorithm on this dataset are linearly estimated based on existing data.

is a factor of 12, and Biased-L2 is a factor of 8 times better than EASE on real-world dataset. Also on this real-world dataset, DRS is a factor of 5 times, and Biased-L2 is a factor of 4 times better than SRS algorithm. On synthetic datasets the differences in accuracy results are slim, but both new algorithms are consistently more accurate than SRS. Looking at the accuracy results in Figure 5, we see that in some datasets, up to 90% accuracy is obtained by using only 3% of the dataset. This result is especially important when mining huge amounts of data. For applications where 90% accuracy is sufficient, instead of running the mining algorithms on the whole data, which can take even days for some datasets, a sample can be used, which is much smaller and easier to handle.

Although the EASE algorithm gives comparable accuracy bounds on synthetic datasets, it performs poorly on the real-world dataset (BMS1), both for sample quality and accuracy, see Figure 4–7 (leftmost). The result is not surprising since the real-world dataset is well known for this kind of behavior, such as, the owners of the dataset claim that most algorithms which work with synthetic datasets do not work well with this real-world dataset [20]. This is the main reason we selected this particular dataset as our real-world control dataset. We want to highlight the fact that Biased-L2 and DRS work perfectly both on synthetic and real-world datasets. On top of this, the major improvements of the present work over EASE are the running time and memory footprint of our new algorithms.

Finally, we compare CPU times for the algorithms we presented in this section. Figure 8 presents the average time spent in milliseconds to process one transaction for each algorithm on a Pentium IV 3Ghz computer. Biased-EA and Biased-L2 are up to 5 times faster compared to EASE. The main reason for this speed-up is the single pass structure of these algorithms compared to logarithmic halving steps in EASE. The CPU time of DRS varies with the sample size, as expected. More

Algorithm/Sampling Rate	0.062	0.03	0.015	0.007
EASE	1.03	1.10	1.13	1.11
Biased-EA	0.21	0.20	0.20	0.20
Biased-L2	0.18	0.20	0.19	0.19
DRS	52.2	21.1	9.6	4.9

Fig. 8. Time spent per transactions (in milliseconds) for each algorithm, with various sampling rates.

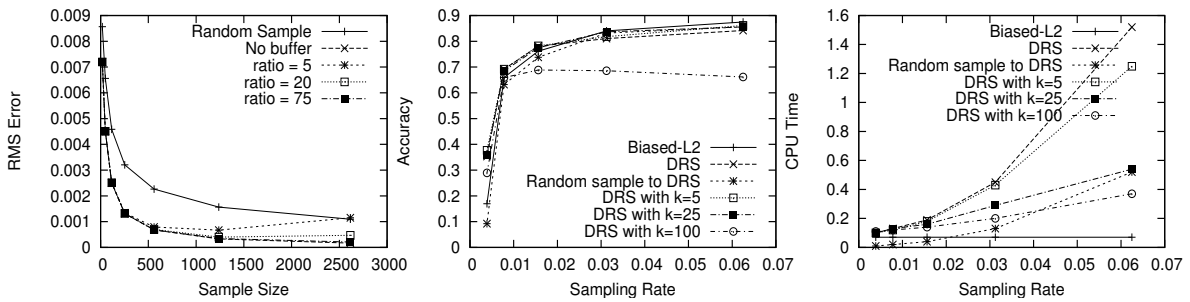


Fig. 9. Effect of  $k$  on sample quality,  $ratio = |S_{DRS}|/k$  (left). Accuracy and CPU time vs. sample rate for Biased-L2 and DRS with different  $k$  values (center, right).

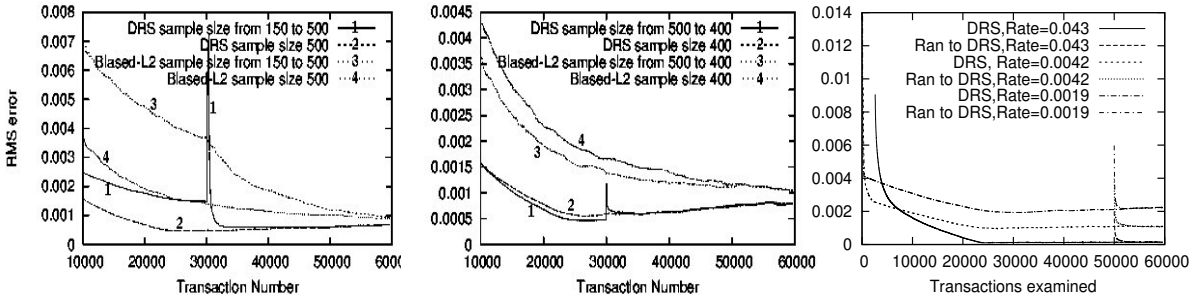


Fig. 10. RMS error vs. number of elements processed, while (left) increasing and (center) decreasing the sample size suddenly after 30 000 transactions. In (right), we compare RMS error while converting random to deterministic samples for different sample sizes.

details about the running time of DRS is presented below in Section "Changing the update rate".

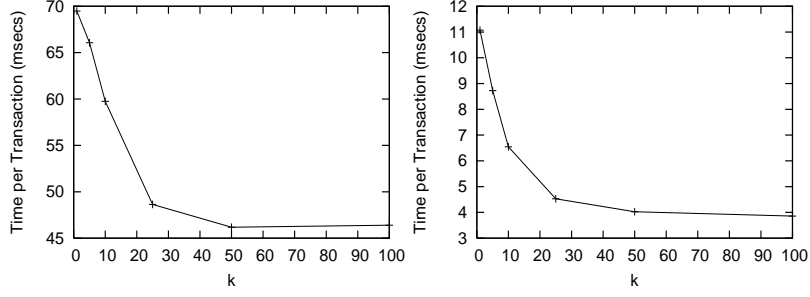


Fig. 11. Effect of changing the  $k$  parameter on the speed of the DRS algorithm. Synthetic dataset (T10I6D100K) (left), and real-world dataset (BMS1) (right). For both datasets selecting  $k \approx 25$  seems quite reasonable. The time spend per transaction on each dataset varies with the average number of items per transaction. The synthetic dataset we test has more items per transaction on average than the real-world (BMS1) dataset, which increases the time per transaction on the synthetic dataset.

### 4.3 Extensions to DRS algorithm

In the previous section we presented the accuracy and quality results of the samples generated by the DRS algorithm. In this section, we further present the additional properties of the DRS algorithm; using  $k$  parameter to control the run-time performance and the fast-recovery property of the algorithm under distribution or sample size changes.

#### **Changing the update rate:**

In Figure 9 (left), the effect of  $k$  on the sample quality of the DRS algorithm is presented. The figure plots the RMS errors of using different values of  $k$  on the BMS1 dataset. As the figure is plotted for different sample sizes, the results are given as the ratio of the sample size over  $k$ . We can see from the figure that the sample quality is similar to a random sample for bigger values of  $k$  (less updates), and quality increases for smaller values of  $k$  (frequent updates). Figure 9 (center

and right) plots the accuracy result and the CPU time of the Biased-L2 and DRS algorithms for various sampling rates. The plots clearly show that the  $k$  parameter in DRS can be used effectively to control the trade-off between the running time of the algorithm and the quality/accuracy of the sample. For example, for sampling rate of 0.062, we can achieve up to a factor of 3 times speed-up in run-time by selecting  $k = 25$ , without sacrificing much from the accuracy. The effect of  $k$  on the runtime of DRS can also be seen in Figure 11. From this figure we can also observe that  $k = 25$  is a reasonable choice, since larger  $k$  values do not significantly decrease the algorithm runtime any further.

### **Changing the sample size:**

In the DRS algorithm we can change the sample size at any time of the sampling process, quite easily. When the sample size is increased from  $s_1$  to  $s_2$ ,  $s_1 < s_2$ , there occurs a gap in the sample of  $(s_2 - s_1)$  transactions. The next transactions from the dataset are added to the sample without any evaluation (in the most basic case). Similarly, when the sample size is decreased from  $s_1$  to  $s_3$ ,  $s_1 > s_3$ , we trim  $(s_1 - s_3)$  transactions out of the sample by finding the worst transaction in the sample and removing it without replacement, enough times. When adding or removing transactions, the item counts of the sample are updated accordingly. After the sample reaches the desired size, it is used as the initial sample for DRS, and the sampling process resumes. Although theoretically it is hard to say how much changing the sample size on the fly affects the quality of the sample, empirical results show that this can cause a major increase in the cost function, but after a very short recovery period, the sample is as good as a deterministic sample again. In other words, once the DRS algorithm starts running again, the cost function decreases dramatically in a very short period of time.

The experiments in Figure 10 are run on the real world dataset BMS1, while processing the transactions in the original order to prevent introducing free randomness in the dataset. In Figure 10 (left), the effect of increasing the sample size is presented. The lower line plots the trace of sampling the BMS1 dataset with a sample size of 500. The upper line plots the trace of sampling the same dataset with a sample size of 150 until the 30 000th transaction, after which the sampling rate is changed to target a final sample size of 500, which causes the jump in the RMS error. After a number of transactions, the RMS error function converges to the value it would get for a sample size of 500. One important point to note here is that, when adding new transactions to the sample we did not use any evaluation criteria, just to demonstrate the effect on the RMS error value. One way to add more transactions is to make the whole process greedy, such that the peak caused by the sample size change would be lower, and the sample would converge gradually. Figure 10 (center) similarly shows the plot of decreasing the sample size from 500 to 400, and shows that the final RMS error value converges to the value it would get for a sample of size 400 (from the first transaction). These results show us that the DRS sample size can be changed at any time during sampling, and the jump in the error function can be compensated for with the RMS error converging to its normal value for the new sampling rate, after examining only a small number of transactions (typically proportional to the size of the sample). Note that the convergence for Biased-L2 is much slower, which clearly shows the better recovery of DRS after a sudden change in sample size.

Another important outcome of the fast convergence of DRS is that we can take a random sample from the dataset at any time, use this sample as initial sample for DRS, and convert this random sample to a new sample after only examining  $O(s)$  new transactions from the dataset, yielding the expected RMS error of a de-

terministic sample for that new sample size. Figure 10 (right) shows the plot of three different sampling processes. The dataset BMS1 is sampled three times with sampling rates of 0.043, 0.0042, and 0.0019 (sample sizes of 2615, 253, and 116 respectively). Also, after examining 50 000 transactions, a simple random sample is created for each case, having exactly the same sizes as 2615, 253, and 116. The plots after transaction 50 000 show the results of using these random samples as initial samples for our algorithm. Clearly, after only examining a small number of new transactions, the RMS errors of the samples converge to the expected value. In the end, it makes little to no difference if we sample the whole dataset one transaction at a time, or if we get a random sample at any time and convert it using our DRS algorithm.

To sum up the experiments in this section, in terms of sample accuracy and quality, both Biased-L2 and DRS outperform EASE and SRS. Biased-L2 is our algorithm of choice if speed is important, and DRS is our choice if either the sample size must be chosen exactly, or if the sampling rate and/or data distribution changes frequently and fast convergence is needed.

## **5 Concluding Remarks**

In this paper, we have presented two novel deterministic sampling algorithms, Biased-L2 and DRS. Both algorithms are designed to sample count data, which is quite common for data mining applications, such as market basket data, web clickstream data and network data. Our new algorithms improve on previous algorithms both in run-time and memory footprint. Furthermore, by conducting extensive simulations on various synthetic and real-world datasets, we have shown that our algorithms generate samples with better accuracy and quality compared to the

previous algorithms (SRS and EASE). Biased-L2 is computationally more efficient than DRS, and when the data is homogeneous or randomly shuffled, produces samples of comparable quality. For sudden changes in the distribution, however, DRS has the ability to remove under- or over-sampled transactions if a more suitable one is found during sampling. In the previous algorithms surveyed and in Biased-L2, transactions added in the early stages of sampling affect the overall quality of the sample especially if the distribution of the dataset changes; DRS is not subject to this limitation.

## References

- [1] R. Agrawal, T. Imielinski and A. Swami. Mining association rules between sets of items in large databases. *Proc. ACM SIGMOD Int. Conf. Management of Data*, pp. 207–216, 1993.
- [2] H. Akcan, H. Brönnimann and R. Marini. Practical and Efficient Geometric Epsilon-Approximations. *Proc. of the 18th Canadian Conference on Computational Geometry*, pp. 121–124, 2006.
- [3] B. Babcock, M. Datar and R. Motwani. Sampling from a moving window over streaming data. *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pp. 633–634, 2002.
- [4] D. Barbará, W. DuMouchel, C. Faloutsos, P. J. Haas, J. M. Hellerstein, Y. E. Ioannidis, H. V. Jagadish, T. Johnson, R. Ng, V. Poosala, K. A. Ross, and K. C. Sevcik. The New Jersey Data Reduction Report. *IEEE Data Engineering Bulletin* 20(4):3–45, 1997.
- [5] H. Brönnimann, B. Chen, M. Dash, P. J. Haas and P. Scheuermann. Efficient data reduction with EASE. *Proc. 9th ACM SIGKDD Int. Conf. Knowledge Discovery & Data Mining (KDD)*, pp. 59–68, 2003.
- [6] H. Brönnimann, B. Chen, M. Dash, P. J. Haas, Y. Qiao and P. Scheuermann. Efficient data-reduction methods for on-line association rule discovery. Chapter 4 of *Selected*

- papers from the NSF Workshop on Next-Generation Data Mining (NGDM'02)*, pp. 190–208, MIT Press, 2004.
- [7] B. Chazelle. *The discrepancy method*. Cambridge University Press, Cambridge, United Kingdom, 2000.
- [8] B. Chen, P. J. Haas and P. Scheuermann. A new two-phase sampling based algorithm for discovering association rules. *Proc. 8th ACM SIGKDD Int. Conf. Knowledge Discovery & Data Mining (KDD)*, pp. 462–468, 2002.
- [9] M. Datar and S. Muthukrishnan. Estimating rarity and similarity on data stream windows. *Proc. ESA*, pp. 323–334, 2002.
- [10] N. Duffield, C. Lund, and M. Thorup. Learn more, sample less: Control of volume and variance in network measurements. *IEEE Transactions on Information Theory*, 51(5): 1756–1775, 2005.
- [11] P. B. Gibbons, S. Acharya, Y. Bartal, Y. Matias, S. Muthukrishnan, V. Poosala, S. Ramaswamy, and T. Suel. Aqua: System and techniques for approximate query answering. Technical report, Bell Labs, 1998.
- [12] P. B. Gibbons, Y. Matias. New sampling-based summary statistics for improving approximate query answers. *Proc. ACM SIGMOD Int. Conf. Management of Data*, pp. 331–342, 1998.
- [13] P. B. Gibbons, Y. Matias, V. Poosala. Fast incremental maintenance of approximate histograms. *Proc. 23rd Int. Conf. Very Large Data Bases (VLDB)*, pp. 466–475, 1997.
- [14] J. Gray, A. Bosworth, A. Layman, H. Pirahesh. Data Cube: A relational aggregation operator generalizing Group-By, Cross-Tab, and Sub-Total. *Proc. 12th Int. Conf. on Data Engineering (ICDE)*, pp. 152–159, 1996.
- [15] S. Guha, N. Koudas, and K. Shim. Approximation and streaming algorithms for histogram construction problems. *ACM Trans. on Database Systems*, Vol. 31, No. 1, pp. 396–438, 2006.
- [16] J. M. Hellerstein, P. J. Haas, and H. Wang. Online aggregation. *Proc. ACM SIGMOD Int. Conf. Management of Data*, pp. 171–182, 1997.

- [17] Intelligent Information Systems. *Synthetic Data Generation Code for Associations and Sequential Patterns*. Research group at the IBM Almaden Research Center. <http://www.almaden.ibm.com/software/quest/Resources/datasets/syndata.html>
- [18] G.H. John and P. Langley. Static versus dynamic sampling for data mining. *Proc. 2nd ACM SIGKDD Int. Conf. Knowledge Discovery & Data Mining (KDD)*, pp. 367-370, 1996.
- [19] T. Johnson, S. Muthukrishnan, I. Rozenbaum. Sampling algorithms in a stream operator. *Proc. ACM SIGMOD Int. Conf. Management of Data*, pp. 1–12, 2005.
- [20] R. Kohavi, C. Brodley, B. Frasca, L. Mason and Z. Zheng. KDD-Cup 2000 organizers' report: Peeling the onion. *SIGKDD Explorations* 2(2):86-98, 2000. <http://www.ecn.purdue.edu/KDDCUP>
- [21] G. Manku and R. Motwani. Approximate frequency counts over data streams. *Proc. 28th Int. Conf. Very Large Data Bases (VLDB)*, pp. 346–357, 2002. *Proc. VLDB'02*, pp. 346–357, 2002.
- [22] F. Olken and D. Rotem. Random sampling from databases: a survey. *Statistics and Computing* 5(1):25-42, March 1995.
- [23] H. Toivonen. Sampling large databases for association rules. *Proc. 22nd Int. Conf. Very Large Data Bases (VLDB)*, pp. 134–145, 1996.
- [24] J. S. Vitter. Random sampling with a reservoir. *ACM Trans. Math. Software* 11(1):37–57, March 1985.
- [25] 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.