On Reservoir Sampling with Deletions

Rainer Gemulla[1], Wolfgang Lehner[1], Peter J. Haas[2]

[1] Technische Universität Dresden, Germany, {gemulla,lehner}@inf.tu-dresden.de
[2] IBM Almaden Research Center, USA, phaas@us.ibm.com

Abstract. Perhaps the most flexible synopsis of a database is a random sample of the data; such samples are widely used to speed up processing of analytic queries and data-mining tasks, enhance query optimization, and facilitate information integration. In this paper, we describe a recently proposed method for incrementally maintaining a uniform random sample of the items in a dataset in the presence of an arbitrary sequence of insertions and deletions. Our scheme, called "random pairing" (RP), maintains a bounded-size uniform sample by using newly inserted data items to compensate for previous deletions. The RP algorithm is the first extension of the almost 40-year-old reservoir sampling algorithm to handle deletions; RP reduces to the "passive" algorithm in [1] when the insertions and deletions correspond to a moving window over a data stream. We also prove that it is not possible to "resize" a bounded-size random sample upwards without accessing the base data.

Keywords: Reservoir sampling, Sample maintenance, Synopsis

1 Introduction

Because of its flexibility, sampling is widely used for quick approximate query answering, statistics estimation, data stream processing, data mining, and data integration. Uniform random sampling, in which all samples of the same size are equally likely, is the most basic of the available sampling schemes. Uniform sampling is ubiquitous in applications: most statistical estimators — as well as the confidence-bound formulas for these estimators — assume an underlying uniform sample. Thus uniformity is a must if it is not known in advance how the sample will be used. In this paper, we show how to maintain a bounded-size sample of a dataset defined by a stream of insertion and deletion transactions. Incremental sample maintenance is a powerful technique, because the abstract notion of the underlying "dataset" can be interpreted very broadly in applications. Indeed, the dataset can actually be an arbitrary view, e.g., over the result of an arbitrary SQL query. Samples over views are particularly good candidates for incremental maintenance, because producing such samples on the fly can require very expensive base-data accesses. The idea is to, in effect, compute the "delta" (set of insertions, updates, and deletions) to the view as the underlying tables are updated and then apply general sample-maintenance methods to the resulting sequence of view modifications.

In the context of a data stream management system (DSMS), often only a subset of the data stream is relevant for query processing. On the one hand, windowing techniques [1] restrict the relevant section of the stream to the most recent elements, where recency is defined either by the position of the elements in the stream or by a timestamp associated with each element. On the other hand, suppose that the stream itself consists of updates to a set of items such as the locations of cars on a highway. Many queries only focus on a certain area of interest, say, a specific section of the highway. In both cases, the scope of the query is continously evolving, that is, items enter and leave. Viewing the query scope as a dataset, each element of the stream can be seen as one or more insertions into or deletions from this dataset. Due to the vast number of data streams and/or the high arrival rate of their elements, it is often necessary to compress the relevant part of the data stream to fit into memory or to reduce processing cost. Again, sampling has proven to be a powerful tool for this type of dataset summarization.

This paper briefly describes a recently proposed method [2] for incrementally maintaining a uniform random sample of an evolving dataset. We also show that it is impossible to "resize" a bounded-size random sample upwards without accessing the base data.

2 Related Work

We consider a (possibly infinite) set $T = \{t_1, t_2, \dots\}$ of unique, distinguishable *items* that are inserted into and deleted from the dataset R over time. As indicated above, we do not require R to be accessible or even materialized. In general, items that are deleted may be subsequently re-inserted. Without loss of generality, we assume throughout that R is initially empty. Thus we consider an infinite sequence of transactions, where each transaction is either an insertion or a deletion of item t_k from R. We restrict attention to "feasible" sequences such that R is a true set and items can only be deleted if they are present in R. Our goal is to ensure that, after each transaction is processed, S is a uniform sample from R.

We limit our attention to bounded-size sampling schemes which do not require access to R at any time.[3] The best known method for incrementally maintaining a sample in the presence of a stream of insertions to the dataset is the classical "reservoir sampling" algorithm [4] (RS), which maintains a simple random sample of a specified size M. The general procedure is as follows: Include the first M items into the sample. For each successive insertion into the dataset, include the inserted item into the sample with probability $M/|R|$, where $|R|$ is the size of the dataset just after the insertion; an included item replaces a randomly selected item in the sample. One deficiency of this method is that it cannot handle deletions, and the most obvious modifications for handling deletions either yield procedures for which the sample size systematically shrinks to 0 over time or which require access to R.

The only known bounded-size sampling scheme which can handle both insertions and deletions has been proposed in [3]. The idea is to include every item into the sample with probability q and to directly remove deleted items from the sample, if present. The sample is then purged every time it exceeds the upper bound, so that the algorithm is best described as Bernoulli sampling with purging (BSP). Starting with $q = 1$, we decrease q at every purge step. With q' being the new value of q, the sample is subsampled using Bernoulli sampling with sampling rate (q'/q). This procedure is repeated until the sample size has fallen below M. The choice of q' is challenging: on the one hand, if q' is chosen small with respect to q, the sample size drops significantly below the upper bound in expectation. On the other hand, a high value of q' leads to frequent purges, thereby reducing performance. Due to the difficulty of choosing q and, as discussed in the sequel, instability in the sample sizes, this algorithm can be difficult to use in practice.

Our new RP algorithm, described in Section 3, maintains a bounded-size uniform sample in the presence of arbitrary insertions and deletions without requiring access to the base data. The RP algorithm produces samples that are significantly larger (i.e., more space efficient) and more stable than those produced by BSP, at lower cost.

3 Random Pairing

To motivate the idea behind the random-pairing scheme, we first consider an "obvious" passive algorithm for maintaining a bounded uniform sample S of a dataset R. The algorithm avoids accessing base data by making use of new insertions to "compensate" for previous deletions. Whenever an item is deleted from the data set, it is also deleted from the sample, if present. Whenever the sample size lies at its upper bound M, the algorithm handles insertions identically to RS; whenever the sample size lies below the upper bound and an item is inserted into the dataset, the item is also inserted into the sample. Although simple, this algorithm is unfortunately incorrect, because it fails to guarantee uniformity. To see this, suppose that, at some stage, $|S| = M < |R| = N$. Also suppose that an item t^- is then deleted from the dataset R, directly followed by an insertion of t^+. Denote by S' the sample after these two operations. If the sample is to be truly uniform, then the probability that $t^+ \in S'$ should equal M/N, conditional on $|S| = M$. Since $t^- \in S$ with

[3] A broader summary of available uniform sampling schemes can be found in [2].

probability M/N, it follows that

$$P\{\, t^+ \in S' \,\} = P\{\, t^- \in S,\ t^+ \text{ incl.} \,\} + P\{\, t^- \notin S,\ t^+ \text{ incl.} \,\} = \frac{M}{N} \cdot 1 + \left(1 - \frac{M}{N}\right) \cdot \frac{M}{N} > \frac{M}{N},$$

conditional on $|S| = M$. Thus an item inserted just after a deletion has an overly high probability of being included in the sample. The basic idea behind RP is to avoid the foregoing problem by including an inserted item into the sample with a probability less than 1 when the sample size lies below the upper bound. The key question is how to select the inclusion probability to ensure uniformity.

3.1 Algorithm Description

In the RP scheme, every deletion from the dataset is eventually compensated by a subsequent insertion. At any given time, there are 0 or more "uncompensated" deletions; the number of uncompensated deletions is simply the difference between the cumulative number of insertions and the cumulative number of deletions. The RP algorithm maintains a counter c_1 that records the number of uncompensated deletions in which the deleted item was in the sample (so that the deletion also decremented the sample size by 1). The RP algorithm also maintains a counter c_2 that records the number of uncompensated deletions in which the deleted item was not in the sample (so that the deletion did not affect the sample). Clearly, $d = c_1 + c_2$ is the total number of uncompensated deletions.

The algorithm works as follows. Deletion of an item is handled by removing the item from the sample, if present, and by incrementing the value of c_1 or c_2, as appropriate. If $d = 0$, i.e., there are no uncompensated deletions, then insertions are processed as in standard RS. If $d > 0$, then we flip a coin at each insertion step, and include the incoming insertion into the sample with probability $c_1/(c_1 + c_2)$; otherwise, we exclude the item from the sample. We then decrease either c_1 or c_2, depending on whether the insertion has been included into the sample or not.

Conceptually, whenever an item is inserted and $d > 0$, the item is paired with a randomly selected uncompensated deletion, called the "partner" deletion. The inserted item is included into the sample if its partner was in the sample at the time of its deletion, and excluded otherwise. The probability that the partner was in the sample is $c_1/(c_1 + c_2)$. For the purpose of the algorithm, it is not necessary to keep track of the identity of the random partner; it suffices to maintain the counters c_1 and c_2. Note that if we repeat the above calculation using RP, we now have $P\{\, t^- \notin S,\ t^+ \text{ included} \,\} - 0$, and we obtain the desired result $P\{\, t^+ \in S' \,\} - M/N$. A correctness proof for the random pairing algorithm is given in [2].

The RP algorithm with $M = 2$ is illustrated in Figure 1 (left). The figure shows all possible states of the sample, along with the probabilities of the various state transitions. The example starts after $i = 2$ items have been inserted into an empty dataset, i.e., the sample coincides with R. The insertion of item t_3 leads to the execution of a standard RS step since there are no uncompensated deletions. This step has three possible outcomes, each equally likely. Next, we remove items t_2 and t_3 from both the dataset and the sample. Thus, at $i = 5$, there are two uncompensated deletions. The insertion of t_4 triggers the execution of a *pairing step*. Item t_4 is conceptually paired with either t_3 or t_2 — these scenarios are denoted by a) and b) respectively — and each of these pairings is equally likely. Thus t_4 compensates its partner, and is included in the sample if and only if the partner was in the sample prior to its deletion. This pairing step amounts to including t_4 with probability $c_1/(c_1 + c_2)$ and excluding t_4 with probability $c_2/(c_1 + c_2)$, where the values of c_1 and c_2 depend on which path is taken through the tree of possibilities. A pairing step is also executed at the insertion of t_5, but this time there is only one uncompensated deletion left: t_2 in scenario a) or t_3 in scenario b). Observe that the sampling scheme is indeed uniform: at each time point, all samples of the same size are equally likely to have been materialized.

Figure 1 (right) displays the time-average sample size for a range of dataset sizes when running RP and BSP (with $q' = 0.8q$). For each dataset size, we used a sequence of insertions to create both the dataset and an initial sample ($M = 100,000$), and then measured changes in the sample size

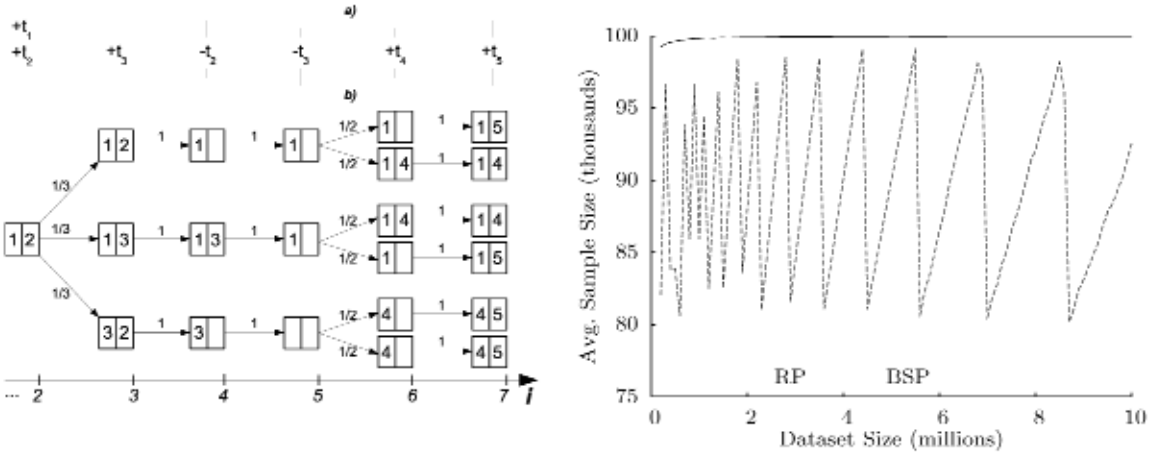

Fig. 1. Random pairing: example and average sample size

as we inserted and deleted 10,000,000 items at random. Clearly, RP produces much more stable samples than BSP, since the latter adjusts the sampling rate only at specific points of time.

3.2 A Negative Result: Resizing Samples Upwards

One might hope that there exist algorithms for resizing a sample upwards without accessing the base data. In general, we consider algorithms that start with a uniform sample S of size at most M from a dataset R and after some finite (possibly zero) number of arbitrary transactions on R — produce a uniform sample S' of size M' from the resulting modified dataset R', where $M < M' < |R|$. Unfortunately, there exists no resizing algorithm that can avoid accessing the base dataset R. To see this, suppose to the contrary that such an algorithm exists, and consider the case in which the transactions on R consist entirely of insertions. Fix a set $A \subseteq R'$ such that $|A| = M'$ and A contains $M + 1$ elements of R; such a set can always be constructed under our assumptions. Because the hypothesized algorithm produces uniform samples of size M' from R', we must have $P\{S' = A\} > 0$. But clearly $P\{S' = A\} = 0$, since $|S| \leq M$ and, by assumption, no further elements of R have been added to the sample. Thus we have a contradiction, and the result follows. A time-optimal resizing algorithm (which may access the base data) is given in [2],

4 Summary

Techniques for incrementally maintaining bounded samples over "datasets" — whether relational tables, views, data streams or other data collections are crucial for unlocking the full power of database sampling techniques. Our new RP algorithm is the algorithm of choice with respect to speed and sample-size stability. We have also shown that it is impossible to resize a bounded sample upwards without accessing base data.

References

1. Brian Babcock, Mayur Datar, and Rajeev Motwani. Sampling from a moving window over streaming data. In *Proc. SODA*, pages 633–634, 2002.
2. Rainer Gemulla, Wolfgang Lehner, and Peter J. Haas. A dip in the reservoir: Maintaining sample synopses of evolving datasets. In *Proc. VLDB*, pages 595 606, 2006.
3. Phillip B. Gibbons and Yossi Matias. New sampling-based summary statistics for improving approximate query answers. In *Proc. ACM SIGMOD*, pages 331–342, 1998.
4. D. E. Knuth. *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*. Addison-Wesley, 1st edition, 1969.