Stream sampling for variance-optimal estimation of subset sums

Edith Cohen[*] Nick Duffield[*] Haim Kaplan[†] Carsten Lund[*] Mikkel Thorup[*]

Abstract

From a high volume stream of weighted items, we want to maintain a generic sample of a certain limited size k that we can later use to estimate the total weight of arbitrary subsets. This is the classic context of on-line reservoir sampling, thinking of the generic sample as a reservoir. We present an efficient reservoir sampling scheme, $\textsc{VarOpt}_k$, that dominates all previous schemes in terms of estimation quality. $\textsc{VarOpt}_k$ provides *variance optimal unbiased estimation of subset sums.* More precisely, if we have seen n items of the stream, then for *any* subset size m, our scheme based on k samples minimizes the average variance over all subsets of size m. In fact, the optimality is against any off-line scheme with k samples tailored for the concrete set of items seen. In addition to optimal average variance, our scheme provides tighter worst-case bounds on the variance of *particular* subsets than previously possible. It is efficient, handling each new item of the stream in $O(\log k)$ time, which is optimal even on the word RAM. Finally, it is particularly well suited for combination of samples from different streams in a distributed setting.

1 Introduction

In this paper we focus on sampling from a high volume stream of weighted items. The items arrive faster and in larger quantities than can be saved, so only a sample can be stored efficiently. We want to maintain a generic sample of a certain limited size that we can later use to estimate the total weight of *arbitrary* subsets.

This is a fundamental and practical problem. In [14] this is the basic function used in a database system for streams. Such a sampling function is now integrated in a measurement system for Internet traffic analysis [8]. In this context, items are records summarizing the flows of IP packets streaming by a router. Queries on selected subsets have numerous current and potential applications, including anomaly detection (detecting unusual traffic patterns by comparing to historic data), traffic engineering and routing (e.g., estimating traffic volume between Autonomous System (AS) pairs), and billing (estimating volume of traffic to or from a certain source or destination). It is important that we are not constrained to subsets known in advance of the measurements. This would preclude exploratory studies, and would not allow a change in routine questions to be applied retroactively to the measurements. A striking example where the selection is not known in advance was the tracing of the *Internet Slammer Worm* [16]. It turned out to have a simple signature in the flow record; namely as being udp traffic to port 1434 with a packet size of 404 bytes. Once this signature was identified, the worm could be studied by selecting records of flows matching this signature from the sampled flow records.

We introduce a new sampling and estimation scheme for streams, denoted $\textsc{VarOpt}_k$, which selects k samples from n items. $\textsc{VarOpt}_k$ has several important qualities: All estimates are *unbiased.* The scheme is *variance optimal* in that it simultaneously minimizes the average variance of weight estimates over subsets of *every* size $m < n$. The average variance optimality is complemented by optimal *worst-case* bounds limiting the variance over all combinations of input streams and queried subsets. These per-subset worst-case bounds are critical for applications requiring robustness and for the derivation of confidence intervals. Furthermore, $\textsc{VarOpt}_k$ is fast. It handles each item in $O(\log k)$ worst-case time, and $O(1)$ expected amortized time for randomly permuted streams.

In Section 5 (Figure 1) we demonstrate the estimation quality of $\textsc{VarOpt}_k$ experimentally via a comparison with other reservoir sampling schemes on the Netflix Prize data set [18]. With our implementation of $\textsc{VarOpt}_k$, the time to sample 1,000 items from a stream of 10,000,000 items was only 7% slower than the time required to read them.

Ignoring the on-line efficiency for streams, there has been several schemes proposed that satisfy the above variance properties both from statistics [2, 25] and indirectly from computer science [21]. Here we formulate the sampling operation $\textsc{VarOpt}_k$ as a general recurrence, allowing independent $\textsc{VarOpt}_k$ samples from different subsets to be naturally combined to obtain a $\textsc{VarOpt}_k$ sample of the entire set. The schemes from [2, 25] fall out as special cases, and we get the flexibility needed for fast on-line reservoir sampling

[*]AT&T Labs—Research, 180 Park Avenue, NJ 07932, USA (email: `(edith,duffield,lund,mthorup)@research.att.com`).

[†]School of Computer Science, Tel Aviv University, Tel Aviv, Israel (email: `haimk@cs.tau.ac.il`)

from a stream. The nature of the recurrence is also perfectly suited for distributed settings.

Below we define the above qualities more precisely and present an elaborate overview of previous work.

1.1 Reservoir sampling with unbiased estimation The problem we consider is classically known as reservoir sampling [15, pp. 138–140]. In reservoir sampling, we process a stream of (weighted) items. The items arrive one at the time, and a reservoir maintains a sample S of the items seen thus far. When a new item arrives, it may be included in the sample S and old items may be dropped from S. Old items outside S are never reconsidered. We think of estimation as an integral part of sampling. Ultimately, we want to use a sample to estimate the total weight of any subset of the items seen so far. Fixing notation, we are dealing with a stream of items where item i has a positive weight w_i. For some integer capacity $k \geq 1$, we maintain a reservoir S with capacity for at most k samples from the items seen thus far. Let $[n] = \{1, \ldots, n\}$ be the set of items seen. With each item $i \in S$ we store a weight estimate $\widehat{w}_i$, which we also refer to as *adjusted weight*. For items $i \in [n] \setminus S$ we have an implicit zero estimate $\widehat{w}_i = 0$. We require these estimators to be unbiased in the sense that $\mathsf{E}[\widehat{w}_i] = w_i$. A typical example is the classic Horvitz-Thompson estimator [13] setting $\widehat{w}_i = w_i / \Pr[i \in S]$ if $i \in S$.

Our purpose is to estimate arbitrary subset sums from the sample. For any subset $I \subseteq [n]$, we let w_I and $\widehat{w}_I$ denote $\sum_{i \in I} w_i$ and $\sum_{i \in I} \widehat{w}_i$, respectively. By linearity of expectation $\mathsf{E}[\widehat{w}_I] = w_I$. Since all unsampled items have 0 estimates, we get $\widehat{w}_{I \cap S} = \widehat{w}_I$. Thus $\widehat{w}_{I \cap S}$, the sum of the adjusted weights of items from the sample that are members of I, is an unbiased estimator of w_I.

Reservoir sampling thus addresses two issues:

- The streaming issue [17] where with limited memory we want to compute a sample from a huge stream that passes by only once.
- The incremental data structure issue of maintaining a sample as new weighted items are inserted. In our case, we use the sample to provide quick estimates of sums over arbitrary subsets of the items seen thus far.

Reservoir versions of different sampling schemes are presented in [3, 5, 10, 12, 11, 27].

1.2 Off-line sampling When considering the qualities of the sample, we compare our on-line scheme, VarOpt_k, with a powerful arbitrary off-line sampling scheme which gets the n weighted items up front, and can tailor the sampling and estimation freely to this concrete set, not having to worry about efficiency or the arrival of more items. The only restriction is the bound k on the number of samples. More abstractly, the off-line sampling scheme is an arbitrary probability distribution Ω over functions $\widehat{w} : [n] \to \mathbb{R}$ from items i to weight estimates $\widehat{w}_i$ which is unbiased in the sense that $\mathsf{E}_{\widehat{w} \leftarrow \Omega}[\widehat{w}_i] = w_i$, and which has at most k non-zeros.

1.3 Statistical properties of target The sampling scheme we want should satisfy some classic goals from statistics. Below we describe these goals. Later we will discuss their relevance to subset sum estimation.

(i) *Inclusion probabilities proportional to size (ipps).* To get k samples, we want each item i to be sampled with probability $p_i = k w_i / w_{[n]}$. This is not possible if some item j has more than a fraction k of the total weight. In that case, the standard is that we include j with probability $p_j = 1$, and recursively ipps sample $k-1$ of the remaining items. The included items are given the standard Horvitz-Thompson estimate $\widehat{w}_i = 1/p_i$.

Note that ipps only considers the marginal distribution on each item, so many joint distributions are possible and in itself, it only leads to an expected number of k items.

(ii) *Sample contains at most k items.* Note that (i) and (ii) together implies that the sample contains exactly $\min\{k, n\}$ items.

(iii) *No positive covariances* between distinct adjusted weights.

From statistics, we know several schemes satisfying the above goals (see, e.g., [2, 25]), but they are not efficient for on-line reservoir sampling. In addition to the above goals, VarOpt_k estimates follow standard Chernoff bounds, and we will elaborate on that in the full version of this paper.

1.4 Average variance optimality Below we will discuss some average variance measures that are automatically optimized by goal (i) and (ii) above.

When n items have arrived, for each subset size $m \leq n$, we consider the average variance for subsets of size $m \leq n$:

$$V_m = \mathsf{E}_{I \subseteq [n], |I| = m} \left[\mathsf{Var}[\widehat{w}_I]\right] = \frac{\sum_{I \subseteq [n], |I| = m} \left[\mathsf{Var}[\widehat{w}_I]\right]}{\binom{n}{m}}.$$

Our VarOpt_k scheme is variance optimal in the following strong sense. For each reservoir size k, stream prefix of n weighted items, and subset size m, there is no off-line sampling scheme with k samples getting a smaller average variance V_m than our generic VarOpt_k.

The average variance measure V_m was introduced in [23] where it was proved that

$$V_m = \frac{m}{n}\left(\frac{n-m}{n-1}\Sigma V + \frac{m-1}{n-1}V\Sigma\right) , \tag{1.1}$$

Here ΣV is the sum of individual variances while $V\Sigma$ is the variance of the estimate of the total, that is,

$$\begin{aligned}\Sigma V &= \sum_{i\in[n]} \mathsf{Var}[\widehat{w}_i] = nV_1\\ V\Sigma &= \mathsf{Var}[\sum_{i\in[n]}\widehat{w}_i] = \mathsf{Var}[\widehat{w}_{[n]}] = V_n.\end{aligned}$$

It follows that we minimize V_m for all m if and only if we simultaneously minimize ΣV and $V\Sigma$, which is what $\textsc{VarOpt}_k$ does. The optimal value for $V\Sigma$ is 0, meaning that the estimate of the total is exact.

Let W_p denote the expected variance of a random subset including each item i independently with some probability p. It is also shown in [23] that $W_p = p\left((1-p)\Sigma V + pV\Sigma\right)$. So if we simultaneously minimize ΣV and $V\Sigma$, we also minimize W_p. It should be noted that both ΣV and $V\Sigma$ are known measures from statistics (see, e.g., [20] and concrete examples in the next section). It is the implications for average variance over subsets that are from [23].

With no information given about which kind of subsets are to be estimated, it makes most sense to optimize average variance measures like those above giving each item equal opportunity to be included in the estimated subset. If the input distributions are not too special, then we expect this to give us the best estimates in practice, using variance as the classic measure for estimation quality.

Relation to goals (i) and (ii) It is standard knowledge from statistics that the ipps from goal (i) are the unique inclusion probabilities that minimize ΣV. (see, e.g., [20, p. 86] for the case of no dominant items, or [10] for the general case). It is also easy to verify that conditioned on (i), goal (ii) is equivalent to $V\Sigma = 0$ (again this appears to be standard, but we couldn't find the general statement, The argument is trivial though. Given the (i), the only variability in the weight estimates returned is in the number of sampled estimates of value τ, so the estimate of the total is variable if and only if the number of samples is variable). The classic goals (i) and (ii) are thus equivalent to minimizing the average variances of this subsection.

1.5 Worst-case robustness In addition to minimizing the average variance, $\textsc{VarOpt}_k$ has some complimentary worst-case robustness properties, limiting the variance for every single (arbitrary) subset. We note that any such bound has to grow with the square of a scaling of the weights. This kind of robustness is important for applications seeking to minimize worst-case vulnerability. The robustness discussed below is all a consequence of the ipps of goal (i) combined with the non-positive covariances of goal (iii).

With the Horvitz-Thompson estimate, the variance of item i is $w_i^2(1/p_i - 1)$. With ipps sampling, $p_i \geq \min\{1, kw_i/w_{[n]}\}$. This gives us the two bounds $\mathsf{Var}[\widehat{w}_i] < w_i w_{[n]}/k$ and $\mathsf{Var}[\widehat{w}_i] < (w_{[n]}/(2k))^2$ (for the second bound note that $p_i < 1$ implies $w_i < w_{[n]}/k$). Both of these bounds are asymptotically tight in that sense that there are instances for which no sampling scheme can get a better leading constant. More precisely, the bound $\mathsf{Var}[\widehat{w}_i] < w_i w_{[n]}/k$ is asymptotically tight if every i has $w_i = o(w_{[n]}/k)$, e.g., when sampling k out of n units, the individual variance we get is $(n/k) - 1$. The bound $(w_{[n]}/(2k))^2$ is tight for $n = 2k$ unit items. In combination with the non-positive covariances of goal (iii), we get that every subset I has weight-bounded variance $\mathsf{Var}[\widehat{w}_I] \leq w_I w_{[n]}/k$, and cardinality bounded variance $\mathsf{Var}[\widehat{w}_I] \leq |I|(w_{[n]}/2k)^2$.

1.6 Efficient for each item With $\textsc{VarOpt}_k$ we can handle each new item of the stream in $O(\log k)$ worst-case time. In a realistic implementation with floating point numbers, we have some precision $\wp$ and accept an error of $2^{-\wp}$. If the stream is viewed as a random permutation of the items, we will show that the expected cost per item is only constant.

In the full version of this paper, for worst-case streams, we will include a matching $\Omega(\log k)$ lower bound on the worst-case time on the word RAM for any floating point implementation of a reservoir sampling scheme with capacity for k samples which minimizes ΣV. Complementing that we will show that it is possible to handle each item in $O(\log\log k)$ amortized time.

1.7 Known sampling schemes We will now discuss known sampling schemes in relation to the qualities of our new proposed scheme:

- Average variance optimality of Section 1.4 following from goal (i) and (ii).
- The robustness of Section 1.5 following from goal (i) and (iii).
- Efficient reservoir sampling implementation with capacity for at most k samples; efficient distributed implementation.

The statistics literature contains many sampling schemes [20, 26] that share some of these qualities, but then they all perform significantly worse on others.

Uniform sampling without replacement In uniform sampling without replacement, we pick a sample of k items uniformly at random. If item i is sampled it gets the Horvitz-Thompson weight estimate $\widehat{w}_i = w_i n/k$. Uniform sampling has obvious variance problems with the frequently-occurring heavy-tailed power-low distributions, where a small fraction of dominant items accounts for a large fraction of the total weight [1, 19], because it is likely to miss the dominant items.

Probability proportional to size sampling with replacement (ppswr) In probability proportional to size sampling (pps) with replacement (wr), each sample $S_j \in [n]$, $j \in [k]$, is independent, and equal to i with probability $w_i/w_{[n]}$. Then i is sampled if $i = S_j$ for some $j \in [k]$. This happens with probability $p_i = 1 - (1 - w_i/w_{[n]})^k$, and if i is sampled, it gets the Horvitz-Thompson estimator $\widehat{w}_i = w_i/p_i$. Other estimators have been proposed, but we always have the same problem with heavy-tailed distributions: if a few dominant items contain most of the total weight, then most samples will be copies of these dominant items. As a result, we are left with comparatively few samples of the remaining items, and few samples imply high variance no matter which estimates we assign.

Probability proportional to size sampling without replacement (ppswor) An obvious improvement to ppswr is to sample without replacement (ppswor). Each new item is then chosen with probability proportional to size among the items not yet in the sample. With ppswor, unlike ppswr, the probability that an item is included in the sample is a complicated function of all the item weights, and therefore the Horvitz-Thompson estimator is not directly applicable. A ppswor reservoir sampling and estimation procedure is, however, presented in [5, 4, 6].

Even though ppswor resolves the "duplicates problem" of ppswr, we claim here a negative result for *any* ppswor estimator: in the full version of this paper, we will present an instance for any sample size k and number of items n such that any estimation based on up to $k + (\ln k)/2$ ppswor samples will perform a factor $\Omega(\log k)$ worse than $\textsc{VarOpt}_k$ for *every* subset size m. This is the first such negative result for the classic ppswor besides the fact that it is not strictly optimal.

Ipps Poisson sampling It is more convenient to think of ipps sampling in terms of a *threshold* τ. We include in the sample S every item with weight $w_i \geq \tau$, using the original weight as estimate $\widehat{w}_i = w_i$. An item i with weight $w_i < \tau$ is included with probability $p_i = w_i/\tau$, and it gets weight estimate τ if sampled.

For an expected number of k samples, we use $\tau = \tau_k$ satisfying

$$\sum_i p_i = \sum_i \min\{1, w_i/\tau_k\} = k. \tag{1.2}$$

This τ_k is unique if $k < n$. For $k \geq n$, we define $\tau_k = 0$ which implies that all items are included. This threshold centric view of ipps sampling is taken from [9].

If the threshold τ is given, and if we are satisfied with Poisson sampling, that is, each item is sampled independently, then we can trivially perform the sampling from a stream. In [10] it is shown how we can adjust the threshold as samples arrive to that we always have a reservoir with an expected number of k samples, satisfying goal (i) for the items seen thus far. Note, however, that we may easily violate goal (ii) of having at most k samples.

Since the items are sampled independently, we have zero covariances, so (iii) is satisfied along with the all the robustness of Section 1.5. However, the average variance of Section 1.4 suffers. More precisely, with zero covariances, we get $V\Sigma = \Sigma V$ instead of $V\Sigma = 0$. From (1.1) we get that for subsets of size m, the average variance is a factor $(n-1)/(n-m)$ larger than for a scheme satisfying both (i) and (ii). Similarly we get that the average variance $W_{\frac{1}{2}}$ over all subsets is larger by a factor 2.

Priority sampling Priority sampling was introduced in [10] as a threshold centric scheme which is tailored for reservoir sampling with k as a hard capacity constraint as in (ii). It is proved in [22] that priority sampling with $k+1$ samples gets as good ΣV as the optimum obtained by (i) with only k samples. Priority sampling has zero covariances like the above ipps Poisson sampling, so it satisfies (iii), but with $V\Sigma = \Sigma V$ it has the same large average variance for larger subsets.

Satisfying the goals but not with efficient reservoir sampling As noted previously, there are several schemes satisfying all our goals [2, 25, 21], but they are not efficient for reservoir sampling or distributed data. Chao's scheme [2] can be seen as a reservoir sampling scheme, but when a new item arrives, it computes all the ipps probabilities from scratch in $O(n)$ time, leading to $O(n^2)$ total time. Tillé [25] has off-line scheme that eliminates items from possibly being in the sample one by one (Tillé also considers a complementary scheme that draws the samples one by one). Each elimination step involves computing elimination probabilities for each remaining item. As such, he ends up spending $O((n-k)n)$ time ($O(kn)$ for the complementary scheme) on selecting k samples. Srinivasan [21] has presented the most efficient off-line scheme, but cast for a different problem. His input are the desired inclusion probabilities p_i that should sum to k. He then selects the k samples in linear time by a simple pairing procedure that can even be used on-line. However, to apply his algorithm to our problem, we first need to compute the ipps probabilities p_i, and to do that, we first need to know all the weights w_i, turning the whole thing into an off-line linear time algorithm. Srinivasan states that he is not aware of any previous scheme that can solve his task, but using his inclusion probabilities, the above mentioned older schemes from statistics [2, 25] will do the job, albeit less efficiently. We shall discuss our technical relation to [2, 25] in more detail in Section 2.3. Our contribution is a scheme VarOpt_k that satisfies all our goals (i)–(iii) while being efficient reservoir sampling from a stream, processing each new item in $O(\log k)$ time.

2 VarOpt_k

By VarOpt_k we will refer to any scheme satisfying our goals (i)–(iii) that we recall below.

(i) Ipps. The sampling probabilities p_i are thus defined via τ_k from (1.2), referred to as *the threshold* when k and the weights are given.

(ii) At most k samples.

(iii) No positive covariances.

Recall that these properties imply all variance qualities mentioned in the introduction. As mentioned in the introduction, our concrete VarOpt_k scheme will also admit standard Chernoff bounds even though these are not implied by (i)–(iii). We will elaborate on this in the full version of this paper.

As mentioned in the introduction, a clean design that differentiates our VarOpt_k scheme from preceding schemes is that we can just sample from samples without relying on auxiliary data. To make sense of this statement, we let all sampling scheme operate on some adjusted weights, which initially are the original weights. When we sample some items with adjusted weight, we use the resulting weight estimates as new adjusted weights, treating them exactly as if they were original weights.

2.1 A general recurrence Our main contribution is a general recurrence. Let $I_1, ..., I_m$ disjoint non-empty sets of weighted items, and $k_1, ..., k_m$ be integers each at least as large as k. Then

$$\text{VarOpt}_k\Big(\bigcup_{x\in[m]} I_x\Big) = \text{VarOpt}_k\Big(\bigcup_{x\in[m]} \text{VarOpt}_{k_x}(I_x)\Big) \tag{2.3}$$

This general recurrence is useful in, say, a distributed setting, where the sets I_x are at different locations and only local samples $\text{VarOpt}_{k_x}(I_x)$ are forwarded to the take part in the global sample.

2.2 Specializing to reservoir sampling To make use of (2.3) in a streaming context, first as a base case, we assume an implementation of $\text{VarOpt}_k(I)$ when I has $k+1$ items, denoting this procedure $\text{VarOpt}_{k,k+1}$. This is very simple and has been done before in [2, 25]. Specializing (2.4) with $m = 2$, $k_1 = k_2 = k$, $I_1 = \{1, ..., n-1\}$ and $I_2 = \{n\}$, we get

$$\text{VarOpt}_k([n]) = \text{VarOpt}_{k,k+1}(\text{VarOpt}_k([n-1]) \cup \{n\}). \tag{2.4}$$

With (2.4) we immediately get a VarOpt_k reservoir sampling algorithm: the first k items fill the initial reservoir. Thereafter, whenever a new item arrives, we add it to the current reservoir sample, which becomes of size $k+1$. Finally we apply $\text{VarOpt}_{k,k+1}$ sample to the result. In the application of $\text{VarOpt}_{k,k+1}$ we do not distinguish between items from the previous reservoir and the new item.

2.3 Relation to Chao's and Tillé's procedures When we use (2.4), we generate exactly the same distribution on samples as that of Chao's procedure [2]. However, Chao does not use adjusted weights. Instead, when a new item n arrives, he computes the new ipps probabilities using the iterative formulation from statics: if some item contains more than a fraction $1/k$ of the total weight, we include it and sample $k-1$ of the remaining items. This iterative computation involves all the original weights even if we are only want the inclusion probability of a given item. Comparing the new and the previous probabilities, he finds the distribution for which item to drop. Our recurrence with adjusted weights is simpler and more efficient because we can forget about the past: the original weights and the inclusion probabilities from previous rounds.

We can also use (2.3) to derive the elimination procedure of Tillé [25]. To do that, we set $m = 1$ and $k_1 = k + 1$, yielding the recurrence

$$\text{VarOpt}_k(I) = \text{VarOpt}_{k,k+1}(\text{VarOpt}_{k+1}(I))$$

This tells us how to draw k samples by eliminating the $n-k$ other items one at the time. Like Chao, Tillé [25] computes the elimination probabilities for all items in all rounds directly from the original weights. Our general recurrence (2.3) based on adjusted weights is more flexible, simpler, and more efficient.

2.4 Relation to previous reservoir sampling schemes It is easy to see that nothing like (2.4) works for any of the other reservoir sampling schemes from the introduction. E.g., if $\textsc{Unif}_k$ denotes uniform sampling of k items with associated estimates, then

$$\textsc{Unif}_k([n]\}) \neq \textsc{Unif}_{k,k+1}(\textsc{Unif}_k([n-1]) \cup \{n\}).$$

With equality, this formula would say that item n should be included with probability $k/(k+1)$. However, to integrate item n correctly in the uniform reservoir sample, we should only include it with probability k/n. The standard algorithms [12, 27] therefore maintain the index n of the last arrival.

We have the same issue with all the other schemes: ppswr, ppswor, priority, and Poisson ipps sampling. For each of these schemes, we have a global description of what the reservoir should look like for a given stream. When a new item arrives, we cannot just treat it like the current items in the reservoir, sampling k out of the $k+1$ items. Instead we need some additional information in order to integrate the new item in a valid reservoir sample of the new expanded stream. In particular, priority sampling [10] and the ppswor schemes of [5, 4, 6] use priorities/ranks for all items in the reservoir, and the reservoir version of Poisson ipps sampling from [9, 10] uses the sum of all weights below the current threshold.

Generalizing from unit weights The standard scheme [12, 27] for sampling k unit items is variance optimal and we can see $\textsc{VarOpt}_k$ as a generalization to weighted items which produces exactly the same sample and estimate distribution when applied to unit weights. The standard scheme for unit items is, of course, much simpler: we include the nth item with probability n/k, pushing out a uniformly random old one. The estimate of any sampled item becomes n/k. With $\textsc{VarOpt}_k$, when the nth item arrives, we have a k old adjusted weights of size $(n-1)/k$ and a new item of weight 1. We apply the general $\textsc{VarOpt}_{k,k+1}$ to get down to k weights. The result is ends up the same: the new item is included with probability $1/n$, and all adjusted weights become n/k.

However, $\textsc{VarOpt}_k$ is not the only natural generalization of the standard scheme for unit weights. The ppswor schemes from [5, 4, 6] also produce the same results when applied to unit weights. However, ppswor and $\textsc{VarOpt}_k$ diverge when the weights are not all the same. The ppswor scheme from [6] does have exact total ($V\Sigma = 0$), but suboptimal ΣV so it is not variance optimal.

Priority sampling is also a generalization in that it produces the same sample distribution when applied to unit weights. However, the estimates vary a bit, and that is why it only optimizes ΣV modulo one extra sample. A bigger caveat is that priority sampling does not get the total exact as it has $V\Sigma = \Sigma V$.

Our $\textsc{VarOpt}_k$scheme is the unique generalization of the standard reservoir sampling scheme for unit weights to general weights that preserves variance optimality.

3 The recurrence

We will now establish the recurrence (2.3). Recall that we have disjoint non-empty sets $I_1, ..., I_m$ of weighted items, and integers $k_1, ..., k_m$ each at least as large as k. We want to prove

$$\textsc{VarOpt}_k(\bigcup_{x\in[m]} I_x) = \textsc{VarOpt}_k(\bigcup_{x\in[m]} \textsc{VarOpt}_{k_x}(I_x))$$

Let $I = \bigcup_{x\in[m]} I_x$. We use w_i to denote the original weights. For each $x \in [m]$, set $I'_x = \textsc{VarOpt}_k(I_x)$, and use w'_i for the resulting adjusted weights. Set $I' = \bigcup_{x\in[m]} I'_x$. Finally, set $S = \textsc{VarOpt}_k(I')$ and use the final adjusted weights as weight estimates $\widehat{w}_i$. Let τ_{x,k_x} be the threshold used in $\textsc{VarOpt}_k(I_x)$, and let τ'_k be the threshold used by $\textsc{VarOpt}_k(I')$. We need to prove that the right hand side of (2.3) is a $\textsc{VarOpt}_k$ scheme of I provided the correctness of each internal use of $\textsc{VarOpt}_k$.

Since an unbiased estimator of an unbiased estimator is an unbiased estimator, it follows that the final $\widehat{w}_i$ are unbiased estimators of the original weights w_i. Since the outer call to $\textsc{VarOpt}_k$ returns at most k samples, we have (ii) satisfied. It remains to prove (i) and (iii). First we consider some trivial degenerate cases.

Lemma 3.1. *(2.3) is satisfied if* $|I'| = \sum_{x\in[m]} |I'_x| \le k$.

Proof. If $|I| = \sum_{x\in[m]} |I_x| \le k$, then there is no active sampling and then we get the full set I on both sides of the equality. Thus we may assume $\sum_{x\in[m]} |I'_x| = \sum_{x\in[m]} \min\{k_x, |I_x|\} \le k < \sum_{x\in[m]} |I|$. This implies that $|I'_x| = k_x \ge k$ for some x. We conclude that $m = 1$, $x = 1$, and $k_1 = k$. Then (2.3) degenerates to the trivial $\textsc{VarOpt}_k(I_1) = \textsc{VarOpt}_k(\textsc{VarOpt}_k(I_1))$. □

In the rest of the proof, we assume $|I'| > k$.

Lemma 3.2. *We have that* $\tau'_k > \tau_{x,k_x}$ *for each* $x \in [m]$.

Proof. Since we have assumed $|I'| > k$, we have $\tau'_k > 0$. The statement is thus trivial for x if $|I_x| \le k$ implying $\tau_{x,k_x} = 0$. However, if $|I_x| \ge k$, then from (i) and (ii) on the call $\textsc{VarOpt}_{k_x}(I_x)$, we get that the returned I'_x has k_x items of weight at least τ_{x,k_x}. These items are all in I'. Since $|I'| > k$, it follows from (1.2) that $\tau'_k > \tau_{x,k_x}$. □

Lemma 3.3. *The final sample S includes all i with $w_i > \tau'_k$. Moreover, each $i \in S$ has $\widehat{w}_i = \max\{w_i, \tau'_k\}$.*

Proof. Since $\tau'_k > \tau_{x,k_x}$, and since (i) hold for each internal $\textsc{VarOpt}_k$ it follows that $i \in I$ has $\widehat{w}_i > \tau'_k$ if and only if $w_i > \tau'_k$. It also follows from the correctness of $\textsc{VarOpt}_k(I')$ that no estimate in the sample can be below τ'_k. □

LEMMA 3.4. *The threshold τ'_k depends only on I.*

Proof. From Lemma 3.3 it follows that the final total estimate is a growing function of τ'_k, but since this total is exact, it follows that τ'_k depends only on I. □

LEMMA 3.5. *The probability that $i \in S$ is $\min\{1, w_i/\tau'_k\}$.*

Proof. Lemma 3.3 gives the result if $w_i \geq \tau'_k$. Suppose $w_i < \tau'_k$. We know from the correctness of $\textsc{VarOpt}_k(I')$ that $\widehat{w}_i = \tau'_k$ if i is in S. Since $\widehat{w}_i$ is unbiased, we conclude that it is included with probability w_i/τ'_k. □

LEMMA 3.6. *τ'_k is equal to the threshold τ_k defined directly for I by (1.2).*

Proof. With p_i the probability that item i is included in S, we know that $\sum p_i = k$ since we ended with k items. Hence by Lemma 3.5, we have $\sum_i \min\{1, w_i/\tau'_k\} = k$. However, we defined τ_k as the unique value such that $\sum_i \min\{1, w_i/\tau_k\} = k$, so we conclude that $\tau'_k = \tau_k$. □

From Lemma 3.3, 3.5, and 3.6, we conclude that (i) is satisfied. Finally, we have to prove (iii).

LEMMA 3.7. *The final sample has no positive covariances as in (iii).*

Proof. Since (iii) is satisfied for each internal call, we know that there are no positive covariances in the adjusted weights w'_i from $I'_x = \textsc{VarOpt}_k(I_x)$. Since these samples are independent, we get no positive covariances in w'_i of all items in $I' = \bigcup_{x\in[m]} I'_x$. Let (I^0, w^0) denote any possible concrete value of (I', w'). Then

$$\begin{aligned}
&\mathsf{E}[\widehat{w}_i\widehat{w}_j] \\
&\quad= \sum_{(I^0,w^0)} \big(\Pr[(I',w') = (I^0,w^0)] \\
&\qquad\qquad \cdot \mathsf{E}[\widehat{w}_i\widehat{w}_j \mid (I',w') = (I^0,w^0)]\big) \\
&\quad\leq \sum_{(I^0,w^0)} \big(\Pr[(I',w') = (I^0,w^0)]\ w^0_i w^0_j\big) \\
&\quad= \mathsf{E}[w'_i w'_j] \ \leq\ w_i w_j.
\end{aligned}$$

□

We have now shown that the sample S we generate satisfies (i), (ii), and (iii), hence that it is a $\textsc{VarOpt}_k$ sample. Thus (2.3) follows.

4 Dropping an item

We will now show how to implement $\textsc{VarOpt}_{k,k+1}$. First we give a basic implementation equivalent to the one used in [2, 25]. Later we will tune our implementation for use on a stream.

The input is a set I of $n = k+1$ items i with adjusted weights $\tilde{w}_i$. We want a $\textsc{VarOpt}_k$ sample of I. First we compute the threshold τ_k such that $\sum_{i\in[n]} \min\{1, \tilde{w}_i/\tau_k\} = k$. We want to include i with probability $p_i = \min\{1, \tilde{w}_i/\tau_k\}$, or equivalently, to drop i with probability $q_i = 1 - p_i$. Here $\sum_{i\in I} q_i = n - k = 1$. We partition the unit interval $[0,1]$ into a segment of size q_i for each i with $q_i > 0$. Finally, we pick a random point $r \in [0,1]$. This hits the interval of some $d \in I$, and then we drop d, setting $S = I \setminus \{d\}$. For each $i \in S$ with $\tilde{w}_i < \tau_k$, we set $\tilde{w}_i = \tau_k$. Finally we return S with these adjusted weights.

LEMMA 4.1. *$\textsc{VarOpt}_{k,k+1}$ is a $\textsc{VarOpt}_k$ scheme.*

Proof. It follows directly from the definition that we use threshold probabilities and estimators, so (i) is satisfied. Since we drop one, we end up with exactly k so (ii) follows. Finally, we need to argue that there are no positive covariances. We could only have positive covariances between items below the threshold whose inclusion probability is below 1. Knowing that one such item is included can only decrease the chance that another is included. Since the always get the same estimate τ_k if included, we conclude that the covariance between these items is negative. This settles (iii). □

4.1 An $O(\log k)$ implementation We will now improve $\textsc{VarOpt}_{k,k+1}$ to handle each new item in $O(\log k)$ time. Instead of starting from scratch, we want to maintain a reservoir with a sample R of size k for the items seen thus far. We denote by R_j the a reservoir after processing item j.

In the appendix we show how to process each item in $O(1)$ expected amortized time if the input stream is randomly permuted.

Consider round $j > k$. Our first goal is to identify the new threshold $\tau = \tau_{k,j} > \tau_{k,j-1}$. Then we subsample k out of the $k+1$ items in $R^{\mathrm{pre}}_j = R_{j-1} \cup \{j\}$. Let $\tilde{w}_{(1)}, ..., \tilde{w}_{(k+1)}$ be the adjusted weights of the items in R^{pre}_j in sorted order, breaking ties arbitrarily. We first identify the largest number t such that $\tilde{w}_{(t)} \leq \tau$. Here

$$\begin{aligned}
\tilde{w}_{(t)} \leq \tau \quad &\Longleftrightarrow \quad k+1-t+(\sum_{x\leq t} \tilde{w}_{(x)})/\tilde{w}_{(t)} \geq k \\
&\Longleftrightarrow \quad (\sum_{x\leq t} \tilde{w}_{(x)})/\tilde{w}_{(t)} \geq t-1\ . \qquad (4.5)
\end{aligned}$$

After finding t we find τ as the solution to

$$(4.6)\ (\sum_{x\leq t}\tilde{w}_{(x)})/\tau = t-1 \iff \tau = (\sum_{x\leq t}\tilde{w}_{(x)})/(t-1)\ .$$

To find the item to leave out, we pick a uniformly random number $r \in (0,1)$, and find the smallest $d \leq t$ such that

$$(4.7)\quad \sum_{x\leq d}(1-\tilde{w}_{(x)}/\tau) \geq r \iff d\tau - \sum_{x\leq d}\tilde{w}_{(x)} \geq r\tau\ .$$

Then the dth smallest item in R_j^{pre}, is the one we drop to create the sample $S = R_j$.

The equations above suggests that we find t, τ, and d by a binary search. When we consider an item during this search we need to know the number of items of smaller adjusted weight, and their total adjusted weight.

To perform this binary search we represent R_{j-1} divided into two sets. The set L of large items with $w_i > \tau_{k,j-1}$ and $\tilde{w}_i = w_i$, and the set $T = R_{j-1} \setminus L$ of small items whose adjusted weight is equal to the threshold $\tau_{k,j-1}$. We represent L in sorted order by a balanced binary search tree. Each node in this tree stores the number of items in its subtree and their total weight. We represent T in sorted order (here in fact the order could be arbitrary) by a balanced binary search tree, where each node in this tree stores the number of items in its subtree. If we multiply the number of items in a subtree of T by $\tau_{k,j-1}$ we get their total adjusted weight.

The height of each of these two trees is $O(\log k)$ so we can insert or delete an element, or concatenate or split a list in $O(\log k)$ time [7]. Furthermore, if we follow a path down from the root of one of these trees to a node v, then by accumulating counters from roots of subtrees hanging to the left of the path, and smaller nodes on the path, we can maintain the number of items in the tree smaller than the one at v, and the total adjusted weight of these items.

We process item j as follows. If item j is large, that is $w_j > \tau_{k,j-1}$, we insert it into the tree representing L. Then we find t by searching the tree over L as follows. While at a node v we compute the total number of items smaller than the one at v by adding to the number of such items in L, $|T|$ or $|T|+1$ depending upon whether $w_j \leq \tau_{k,j-1}$ or not. Similarly, we compute the total adjusted weight of items smaller than the one at v by adding $|T|\tau_{k,j-1}$ to the total weight of such items L, and w_j if $w_j \leq \tau_{k,j-1}$. Then we use Equation (4.5) to decide if t is the index of the item at v, or we should proceed to the left or to the right child of v. After computing t we compute τ by Equation (4.6). Next we identify d by first considering item j if $w_j < \tau_{k,j-1}$, and then searching either the tree over T or the tree over L in a way similar to the search for computing t but using Equation (4.7). Once finding d our subsample becomes $R_j = S = R_j^{\text{pre}} \setminus \{d\}$. All this takes $O(\log k)$.

Last we update our representation of the reservoir, so that it corresponds to R_j and $\tau_{k,j}$. We insert w_j into T if $w_j \leq \tau_{k,j-1}$ (otherwise it had already been inserted into L). We also delete d from the list containing it. If $w_{(t)}$ was a large weight we split L at $w_{(t)}$ and concatenate the prefix of L to T. Our balanced trees support concatenation and split in $O(\log k)$ time, so this does not affect our overall time bounds. Thus we have proved the following theorem.

Theorem 4.1. *With the above implementation, our reservoir sampling algorithm processes each new item in $O(\log k)$ time.*

In the above implementation we have assumed constant time access to real numbers including the random $r \in (0,1)$. Real computers do not support real reals, so in practice we would suggest using floating point numbers with precision $\wp \gg \log n$, accepting a fractional error of order $1/2^{\wp}$.

We do have an alternative implementation based on a standard priority queue, but it is only efficient in the amortized sense. Using the priority queue from [24], it handles k items in $O(k \log\log k)$ time. For space reasons, we defer this alternative to the journal version of this paper.

4.2 Faster on randomly permuted streams We will now discuss some faster implementations in amortized and randomized settings. First we consider the case where the input stream is viewed as randomly permuted.

We call the processing of a new item *simple* if it is not selected for the reservoir and if the threshold does not increase above any of the previous large weights. We will argue that the simple case is dominating if $n \gg k$ and the input stream is a random permutation of the weights. Later we get a substantial speed-up by reducing the processing time of the simple case to a constant.

Lemma 3.5 implies that our reservoir sampling scheme satisfies the condition of the following simple lemma:

Lemma 4.2. *Consider a reservoir sampling scheme with capacity k such that when any stream prefix I has passed by, the probability that $i \in I$ is in the current reservoir is independent of the order of I. If a stream of n items is randomly permuted, then the expected number of times that the newest item is included in the reservoir is bounded by $k(\ln(n/k) + O(1))$.*

Proof. Consider any prefix I of the stream. The average probability that an item $i \in I$ is in the reservoir R is $|R|/|I| \leq k/|I|$. If I is randomly permuted, then this is the expected probability that the last item of I is in R. By linearity of expectation, we get that the expected number of times the newest item is included in R is bounded by $k + \sum_{j=k+1}^{n} k/j = k(1 + H_n - H_{k+1}) = k(\ln(n/k) + O(1))$. □

As an easy consequence, we get

Lemma 4.3. *When we apply our reservoir sampling algorithm to a randomly permuted stream, the expected number of times that the threshold passes a weight in the reservoir is bounded by $k(\ln(n/k) + O(1))$.*

Proof. Since the threshold is increasing, a weight in the reservoir can only be passed once, and we know from Lemma 4.2 that the expected number of weights ever entering the reservoir is bounded by $k(\ln(n/k) + O(1))$. □

We now show how to perform a simple case in constant time. To do so, we maintain the smallest of the large weights in the reservoir in a variable w_ℓ.

We now start the processing of item j, hoping for it to be a simple case. We assume we know the cardinality of the set T of items in R_{j-1} with weight no higher than $\tau_{k,j-1}$. Tentatively as in (4.6) we compute

$$\tau = (w_j + |T|\tau_{k,j-1})/|T|.$$

If $w_j \geq \tau$ or $\tau \geq w_\ell$, we cannot be in the simple case, so we revert to the original implementation. Otherwise, τ has its correct value, and we proceed to generate the random number $r \in (0, 1)$ from the original algorithm. If

$$(\tau - w_j) > r\tau,$$

we would include the new item, so we revert to the original algorithm using this value of r. Otherwise, we skip item j setting $\tau_{k,j} = \tau$. No further processing is required, so we are done in constant time. The reservoir and its division into large and small items is unchanged.

Theorem 4.2. *A randomly permuted stream of length n is processed in $O(n + k(\log k)(\log n))$ time.*

Proof. We spend only constant time in the simple cases. From Lemma 4.2 and 4.3 we get that the expected number of non-simple cases is at most $2k(\ln(n/k) + O(1)) = O(k(\log(n/k))$, and we spend only $O(\log k)$ time in these cases. □

5 Some experimental results on Netflix data

We illustrate both the usage and the estimate quality attained by VarOpt$_k$ through an example on a large real-life data set. The Netflix Prize [18] data set consists of reviews of 17,770 distinct movie titles by 5×10^5 reviewers. The weight we assigned to each movie title is the corresponding number of reviews. We experimentally compare VarOpt to state of the art reservoir sampling methods. All methods produce a fixed-size sample of $k = 1000$ titles along with an assignment of adjusted weights to included titles. These summaries (titles and adjusted weights) support unbiased estimates on the weight of subpopulations of titles specified by arbitrary selection predicate. Example selection predicates are "PG-13" titles, "single-word" titles, or "titles released in the 1920's". An estimate of the total number of reviews of a subpopulation is obtained by applying the selection predicate to all titles included in the sample and summing the adjusted weights over titles for which the predicate holds.

We partitioned the titles into subpopulations and computed the sum of the square errors of the estimator over the partition. We used natural set of partitions based on ranges of release-years of the titles (range sizes of 1,2,5,10 years). Specifically, for partition with range size r, a title with release year y was mapped into a subset containing all titles whose release year is y mod r. We also used the value $r = 0$ for single-titles (the finest partition).

The methods compared are priority sampling (pri) [10], ppswor (probability proportional to size sampling with replacement) with the rank-conditioning estimator (ws RC) [4, 6], ppswor with the subset-conditioning estimator (ws SC) [4, 6], and VarOpt. We note that ws SC dominates (has smaller variance on all distributions and subpopulations) ws RC, which in turn, dominates the classic ppswr Horvitz-Thomson estimator [4, 6]. Results are shown in Figure 1.

The pri and ws RC estimators have zero covariances, and therefore, as Figure 1 shows[1], the sum of square errors is invariant to the partition (the sum of variances is equal to ΣV).

The ws SC and ws RC estimators have the same ΣV and pri [22] has nearly the same ΣV as the optimal VarOpt. Therefore, as the figure shows, on single-titles ($r = 0$), ws RC performs the same as ws SC and pri performs (essentially) as well as VarOpt. Since VarOpt has optimal (minimal) ΣV, it outperforms all other algorithms.

We next turn to larger subpopulations. Figure 1

[1]The slight increase disappears as we average over more and more runs.

illustrates that for VarOpt and the ws SC, the sum of square errors *decreases* with subpopulation size and therefore they have significant benefit over pri and ws RC. We can see that VarOpt, that has optimal average variance for any subpopulation size outperforms ws SC.

To conclude, VarOpt is the winner, being strictly better than both pri and ws SC. We do have theoretical examples where VarOpt$_k$ has a variance that is a factor $\Omega(\log k)$ smaller than that of *any* ppswor scheme, ws SC included, so the performance gains of VarOpt$_k$ can be much larger than on this particular real-life data set.

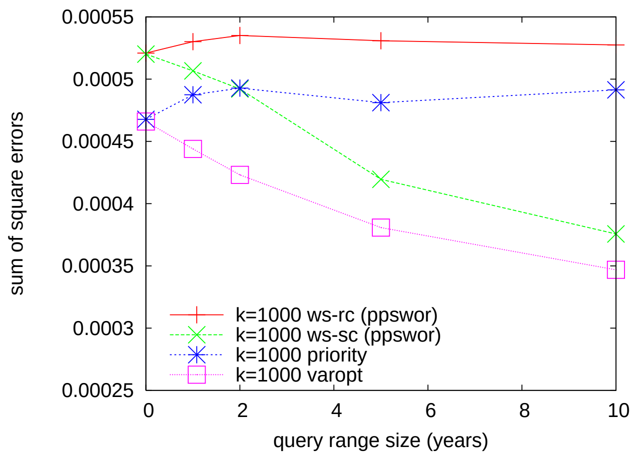

Figure 1: Sum of the square errors of the estimates over each partition, averaged over 500 repetitions of the respective summarization method.

References

[1] R.J Adler, R.E. Feldman, and M.S. Taqqu. *A Practical Guide to Heavy Tails*. Birkhauser, 1998.

[2] M. T. Chao. A general purpose unequal probability sampling plan. *Biometrika*, 69(3):653–656, 1982.

[3] S. Chaudhuri, R. Motwani, and V.R. Narasayya. On random sampling over joins. In *Proc. ACM SIGMOD Conference*, pages 263–274, 1999.

[4] E. Cohen and H. Kaplan. Bottom-k sketches: Better and more efficient estimation of aggregates (poster). In *Proc. ACM SIGMETRICS/Performance*, pages 353–354, 2007.

[5] E. Cohen and H. Kaplan. Summarizing data using bottom-k sketches. In *Proc. 26th ACM PODC*, 2007.

[6] E. Cohen and H. Kaplan. Tighter estimation using bottom-k sketches. In *Proceedings of the 34th VLDB Conference*, 2008.

[7] Th. H. Cormen, Ch. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms*. MIT Press, McGraw-Hill, 2nd edition, 2001.

[8] C. Cranor, T. Johnson, V. Shkapenyuk, and O. Spatcheck. Gigascope: A stream database for network applications. In *Proc. ACM SIGMOD*, 2003.

[9] N.G. 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.

[10] N.G. Duffield, C. Lund, and M. Thorup. Priority sampling for estimation of arbitrary subset sums. *J. ACM*, 54(6):Article 32, December, 2007. Announced at SIGMETRICS'04.

[11] P. S. Efraimidis and P. G. Spirakis. Weighted random sampling with a reservoir. *Inf. Process. Lett.*, 97(5):181–185, 2006.

[12] C.T. Fan, M.E. Muller, and I. Rezucha. Development of sampling plans by using sequential (item by item) selection techniques and digital computers. *J. Amer. Stat. Assoc.*, 57:387–402, 1962.

[13] D. G. Horvitz and D. J. Thompson. A generalization of sampling without replacement from a finite universe. *J. Amer. Stat. Assoc.*, 47(260):663–685, 1952.

[14] T. Johnson, S. Muthukrishnan, and I. Rozenbaum. Sampling algorithms in a stream operator. In *Proc. ACM SIGMOD*, pages 1–12, 2005.

[15] D.E. Knuth. *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms*. Addison-Wesley, 1969.

[16] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver. Inside the slammer worm. *IEEE Security and Privacy Magazine*, 1(4):33–39, 2003.

[17] S. Muthukrishnan. Data streams: Algorithms and applications. *Foundations and Trends in Theoretical Computer Science*, 1(2), 2005.

[18] The Netflix Prize. `http://www.netflixprize.com/`.

[19] K. Park, G. Kim, and M. Crovella. On the relationship between file sizes, transport protocols, and self-similar network traffic. In *Proc. 4th IEEE Int. Conf. Network Protocols (ICNP)*, 1996.

[20] C-E. Särndal, B. Swensson, and J. Wretman. *Model Assisted Survey Sampling*. Springer, 1992.

[21] A. Srinivasan. Distributions on level-sets with applications to approximation algorithms. In *Proc. 41st FOCS*, pages 588–597. IEEE, 2001.

[22] M. Szegedy. The DLT priority sampling is essentially optimal. In *Proc. 38th STOC*, pages 150–158, 2006.

[23] M. Szegedy and M. Thorup. On the variance of subset sum estimation. In *Proc. 15th ESA, LNCS 4698*, pages 75–86, 2007.

[24] M. Thorup. Equivalence between priority queues and sorting. *J. ACM*, 54(6):Article 28, December, 2007. Announced at FOCS'02.

[25] Y. Tillé. An elimination procedure for unequal probability sampling without replacement. *Biometrika*, 83(1):238–241, 1996.

[26] Y. Tillé. *Sampling Algorithms*. 2006.

[27] J.S. Vitter. Random sampling with a reservoir. *ACM Trans. Math. Softw.*, 11(1):37–57, 1985.