Journal of Computer and System Sciences 78 (2012) 260–272

Contents lists available at ScienceDirect

Journal of Computer and System Sciences

www.elsevier.com/locate/jcss

Optimal sampling from sliding windows

Vladimir Braverman [a,*], Rafail Ostrovsky [b], Carlo Zaniolo [c]

[a] 3771 Boelter Hall, UCLA, Computer Science Department, Los Angeles, CA 90095, United States
[b] 3732D Boelter Hall, UCLA, Computer Science Department, Los Angeles, CA 90095, United States
[c] 3532G Boelter Hall, UCLA, Computer Science Department, Los Angeles, CA 90095, United States

ARTICLE INFO

Article history:
Received 26 May 2009
Received in revised form 7 April 2011
Accepted 20 April 2011
Available online 28 April 2011

Keywords:
Data streams
Random sampling
Sliding windows

ABSTRACT

A *sliding windows* model is an important case of the streaming model, where only the most "recent" elements remain active and the rest are discarded. The sliding windows model is important for many applications (see, e.g., Babcock, Babu, Datar, Motwani and Widom (PODS 02); and Datar, Gionis, Indyk and Motwani (SODA 02)). There are two equally important types of the sliding windows model – windows with fixed size (e.g., where items arrive one at a time, and only the most recent n items remain active for some fixed parameter n), and timestamp-based windows (e.g., where many items can arrive in "bursts" at a single step and where only items from the last t steps remain active, again for some fixed parameter t). *Random sampling* is a fundamental tool for data streams, as numerous algorithms operate on the sampled data instead of on the entire stream. Effective sampling from sliding windows is a nontrivial problem, as elements eventually expire. In fact, the deletions are *implicit*; i.e., it is not possible to identify deleted elements without storing the entire window. The implicit nature of deletions on sliding windows does not allow the existing methods (even those that support explicit deletions, e.g., Cormode, Muthukrishnan and Rozenbaum (VLDB 05); Frahling, Indyk and Sohler (SOCG 05)) to be directly "translated" to the sliding windows model. One trivial approach to overcoming the problem of implicit deletions is that of over-sampling. When k samples are required, the over-sampling method maintains $k' > k$ samples in the hope that at least k samples are not expired. The obvious disadvantages of this method are twofold:

(a) It introduces additional costs and thus decreases the performance; and
(b) The memory bounds are not deterministic, which is atypical for streaming algorithms (where even small probability events may eventually happen for a stream that is long enough).

Babcock, Datar and Motwani (SODA 02), were the first to stress the importance of improvements to over-sampling. They formally introduced the problem of sampling from sliding windows and improved the over-sampling method for *sampling with replacement*. Their elegant solutions for sampling with replacement are optimal *in expectation*, and thus resolve disadvantage (a) mentioned above. Unfortunately, the randomized bounds do not resolve disadvantage (b) above. Interestingly, all algorithms that employ the ideas of Babcock, Datar and Motwani have the same central problem of having to deal with a randomized complexity bound (see, e.g., Datar and Muthukrishnan (ESA 02); Chakrabarti, Cormode and McGregor (SODA 07)). Further, the proposed solutions of Babcock, Datar and Motwani for *sampling without replacement* are based on the over-sampling method and thus do not solve problem (a). Therefore, the question of whether we can solve sampling on

* Corresponding author. Fax: +1 (310) 825 7578.
E-mail addresses: vova@cs.ucla.edu (V. Braverman), rafail@cs.ucla.edu (R. Ostrovsky), zaniolo@cs.ucla.edu (C. Zaniolo).
URLs: http://www.cs.ucla.edu/~vova (V. Braverman), http://www.cs.ucla.edu/~rafail (R. Ostrovsky), http://www.cs.ucla.edu/~zaniolo (C. Zaniolo).

0022-0000/$ – see front matter
doi:10.1016/j.jcss.2011.04.004

sliding windows optimally (i.e., resolve both disadvantages) is implicit in the paper of Babcock, Datar and Motwani and has remained open for all variants of the problem. In this paper we answer these questions affirmatively and provide optimal sampling schemas for all variants of the problem, i.e., sampling with or without replacement from fixed or timestamp-based windows. Specifically, for fixed-size windows, we provide optimal solutions that require $O(k)$ memory; for timestamp-based windows, we show algorithms that require $O(k \log n)$ space,[1] which is optimal since it matches the lower bound by Gemulla and Lehner (SIGMOD 08). In contrast to the work of Babcock, Datar and Motwani, our solutions have deterministic bounds.

1. Introduction

Random sampling and sliding windows are two fundamental concepts for data streams. Sampling is a very natural way to summarize data properties with sublinear space; indeed, it is a key component of many streaming algorithms and techniques. Just to mention a few, the relevant papers include Aggarwal [2]; Alon, Duffield, Lund and Thorup [3]; Alon, Matias and Szegedy [4]; Babcock, Babu, Datar, Motwani and Widom [8]; Babcock, Datar and Motwani [10]; Bar-Yossef [13]; Bar-Yossef, Kumar and Sivakumar [16]; Buriol, Frahling, Leonardi, Marchetti-Spaccamela and Sohler [19]; Chakrabarti, Cormode and McGregor [20]; Chaudhuri and Mishra [23]; Chaudhuri, Motwani and Narasayya [24]; Cohen [25]; Cohen and Kaplan [26]; Cormode, Muthukrishnan and Rozenbaum [27]; Dasgupta, Drineas, Harb, Kumar and Mahoney [30]; Datar and Muthukrishnan [32]; Duffield, Lund and Thorup [33]; Frahling, Indyk and Sohler [36]; Gandhi, Suri and Welzl [38]; Gemulla [39]; Gemulla and Lehner [40]; Gibbons and Matias [41]; Guha, Meyerson, Mishra, Motwani and O'Callaghan [46]; Haas [47]; Kolonko and Wäsch [50]; Li [53]; Palmer and Faloutsos [56]; Szegedy [58]; and Vitter [60]. These papers illustrate the vitality of effective sampling methods for data streams. Among other methods, *uniform random sampling* is the most general and well understood. Most applications maintain multiple samples using two popular methods: namely, *sampling with replacement* and *sampling without replacement*. The former method assumes independence among samples; the latter forbids repetitions. While sampling without replacement preserves more information, sampling with replacement is sometimes preferred due to its simplicity; thus both schemas are important for applications.

The concept of sliding windows expresses the importance of recent data for applications. In this model, analysis is restricted to the most recent portion of the stream; the outdated elements must not be considered. The importance of the sliding windows model is well illustrated by the considerable amount of relevant papers in both theory and database communities. A small sample subset of relevant papers includes the work of Arasu, Babcock, Babu, Cieslewicz, Datar, Ito, Motwani, Srivastava and Widom [5]; Arasu and Manku [6]; Ayad and Naughton [7]; Babcock, Babu, Datar, Motwani and Thomas [9]; Babcock, Babu, Datar, Motwani and Widom [8]; Babcock, Datar and Motwani [10,11]; Babcock, Datar, Motwani and O'Callaghan [12]; Das, Gehrke and Riedewald [29]; Datar, Gionis, Indyk and Motwani [31]; Datar and Motwani, Chapter 8 [1]; Datar and Muthukrishnan [32]; Feigenbaum, Kannan and Zhang [34]; Gibbons and Tirthapura [42]; Golab, DeHaan, Demaine, Lopez-Ortiz and Munro [43]; Golab and Özsu [44]; Lee and Ting [52]; Li, Maier, Tufte, Papadimos and Tucker [54]; and Tatbul and Zdonik [59].

Two types of sliding windows are widely recognized. *Fixed-size* (or *sequence-based*) windows define a fixed amount of the most recent elements to be active. For instance an application may restrict an analysis to the last trillion elements. Fixed-size windows are important for applications where the arrival rate of the data is fixed (but still extremely fast), such as sensors or stock market measurements. In *timestamp-based* windows the validity of an element is defined by an additional parameter such as a timestamp. For instance, an application may restrict an analysis to elements that arrived within the last hour. Timestamp-based windows are important for applications with asynchronous data arrivals, such as networking or database applications.

The importance of both concepts raises two natural questions of *optimal sampling from sliding windows*:

Question 1.1. How to maintain a uniform random sampling from sliding windows using provably optimal memory bounds?

Question 1.2. Is sampling from sliding windows algorithmically harder than sampling from the entire stream?

In this paper we answer both questions. Informally, what we show is that it is possible to "translate" (with optimal deterministic memory bounds for all sampling-based algorithms) sampling with and without replacement on the entire stream to sampling with or without replacement in all variants of the sliding windows model. We state precise results in Theorems 2.1, 2.2, 3.9, 4.4 and 5.1.

[1] Here and henceforth we use n to denote the size of the sliding window, i.e., the number of active elements.

1.1. Discussion and related work

In spite of their apparent simplicity, both Questions 1.1 and 1.2 have no trivial solution. Indeed, the sliding windows model implies eventual deletions of samples; thus, none of the well-known methods for insertion-only streams (such as the reservoir method [60]) is applicable. Moreover, the deletions are *implicit*, i.e., they are not triggered by an explicit user's request. Thus, the algorithms for streams with explicit deletions (such as [27,36]) do not help. Let us illustrate the inherent difficulty of sampling from sliding windows with the following example. Consider the problem of maintaining a single sample from a window of the last 2^{30} elements. Assume that the 100-th element is picked as an initial sample. Eventually, the $(2^{30}+100)$-th element arrives, in which case the sample is outdated. But at this time, the data has already been passed and cannot be sampled.

Babcock, Datar and Motwani [10] were the first to address the problem of sampling from sliding windows. The key idea of the elegant algorithms in [10] is a "successors list"; in fact, this idea has been used in almost all subsequent papers. The successors list method suggests backing up a sample with a list of active successors. When a sample expires, the next successor in the list becomes a sample; thus a sample is available at any moment. Based on this idea, Babcock, Datar and Motwani built solutions for sampling with replacement. For sequence-based windows of size n, their *chain sampling* algorithm picks a successor among n future elements, and stores it as it arrives. They show that such schema has an expected memory bound of $O(k)$ and with high probability will not exceed $O(k\log n)$. For timestamp-based windows, their *priority sampling* method associates a priority with every new element. A priority is a random number from $(0,1)$; a sample is an element p with highest priority; and a sample's successor is an element with the highest priority among all elements that arrived after p. Priority sampling requires $O(k\log n)$ memory words in expectation and with high probability. There is a lower bound of $\Omega(k\log n)$ for timestamp-based windows that was shown in [40]. Thus, the methods of Babcock, Datar and Motwani are *optimal in expectation*. However, the inherent problem of the replacement method is that the size of the successors list is itself a random variable; thus this method cannot provide *worst-case optimal* bounds. Moreover, Babcock, Datar and Motwani suggested over-sampling as a solution for sampling without replacement; thus the problem of further improvements is implicitly present in their paper. In his excellent survey, Haas [47] gave a further detailed discussion of their solutions.

Zhang, Li, Yu, Wang and Jiang [61] provide an adaptation of reservoir sampling to sliding windows. However, their approach requires storing the window in memory; thus it is applicable only for small windows. In an important work, Gemulla and Lehner [40] addressed the question of sampling without replacement for timestamp-based windows. They suggest a natural extension of priority sampling by maintaining a list of elements with the k highest priorities. This gives an expected $O(k\log n)$ solution. However, their memory bounds are still randomized. Gemulla [39] and Gemulla and Lehner [40] recently addressed the problem of random sampling from timestamp-based windows with a bounded memory. This setting is different from the original problem of Babcock, Datar and Motwani [10]. Namely, it introduces additional uncertainty in the following sense: there is no guarantee that a sample is available at any moment. They provide a "local" lower bound on the success probability that depends on the window's data. However, there is no "global" lower bound; as Gemulla [39] states in his thesis:

> *We cannot guarantee a global lower bound other than 0 that holds at any arbitrary time without a priori knowledge of the data stream.*

Thus, the problem of optimal sampling for the entire life-span of the stream from sliding windows remained an open problem for all versions until today. We stress that, while this problem is important in its own right, it also has further implications for many other problems. Indeed, uniform random sampling is a key tool for many streaming problems (see, e.g., [55]). "Translations" to sliding windows using previous methods introduce randomized complexity instead of deterministic memory bounds (see, e.g., [20]).

1.2. Our contribution

In this paper we answer affirmatively to Questions 1.1 and 1.2 for all variants of the problem, i.e., for sampling with and without replacement from fixed-size or timestamp-based windows. Our solutions have provable optimal memory guarantees and are stated precisely in Theorems 2.1, 2.2, 3.9, 4.4 and 5.1. In particular, we give $O(k)$ bounds for fixed-size windows (for sampling with or without replacement) and $O(k\log n)$ bounds for timestamp-based windows (for sampling with or without replacement). This is a strict improvement over previous methods that closes the gap between randomized and deterministic complexity, an important fact in its own right.

Finally, this paper introduces simple (yet novel) techniques that are different from all previous approaches. In particular, we reduce the problem of sampling without replacement to the problem of sampling with replacement for all variants of the sliding windows model. This may be of independent interest, since the former method is a more general case than the latter; thus our paper also proves equivalence for sliding windows, as discussed in the next section.

1.3. High-level ideas of our approach

We start by describing four key ideas: *equivalent-width partitions*, *covering decomposition*, *generating implicit events*, and *black-box reduction from sampling without replacement to sampling with replacement*. Also, we outline our approach by giving high-level descriptions of the most important steps.

1.3.1. Equivalent-width partitions

Our methods for the sequence-based windows are based on a simple (yet novel) idea of *equivalent-width partitions*: consider sets A, B, C such that $C \subseteq B \subseteq A \cup C$ and $A \cap C = \emptyset$ and $|B| = |A|$. Our goal is to obtain a sample from B, given samples from A and C. We use the following rule: if a sample from A belongs to B, then we assign it to be a sample from B; otherwise, we assign a sample from C to be a sample from B. The direct computations show the correctness of this schema, i.e., the result is always a uniform sample from B.

As a next step, observe that the above idea can be applied to the sliding windows model. We partition (logically) the entire stream into disjoint intervals (we call them *buckets*) of size n, where n is the size of the window. For each bucket we maintain a random sample using any one-pass algorithm (e.g., the reservoir sampling method). If the window coincides with the most recent bucket, then our task is easy; we assign this bucket's sample to be the output. Otherwise, the window intersects the two most recent buckets. It must be the case that the most recent bucket is "partial"; i.e., not all elements have arrived yet. But this case matches precisely our key idea: the most recent bucket corresponds to C, our window corresponds to B and the second-most recent bucket corresponds to A. We thus can apply the above rule and obtain our sample using only samples from the two buckets. Since we need only these samples, the optimality of our schema is straightforward.

The above idea can be generalized to sampling without replacement. Indeed, we show that, given k-samples without replacement from A and C, we take the portion of A's sample that belongs to B and complete it with the random portion from C's sample. We show that the result is a k-sample without replacement from B. As before, we apply this idea to sliding windows; the detailed proofs can be found in the main body of this paper.

1.3.2. Covering decomposition and generating implicit events

For timestamp-based windows, the size of the window is unknown; moreover, it was shown (see, e.g., [31]) that the size of the window cannot be computed precisely with sublinear memory. This negative result is a key difference between timestamp-based windows and all other models, such as insertion-only streams and streams with explicit deletions (the turnstile model). In fact, this negative result is one of the main reasons for the randomized bounds in previous solutions. Indeed, it is not clear at all how to obtain uniformity if even the size of the sampled domain is unknown.

Our key observation is that it is possible to sample from a window without knowledge of its size. As before, consider disjoint sets A, B, C such that $C \subseteq B \subseteq A \cup C$ and $A \cap C = \emptyset$. In the current scenario we do not assume that $|A| = |B|$ and still obtain samples from B. We show that if $|A| \leqslant |B|$, and it is possible to generate random events w.p. $\frac{|A|}{|B|}$, then it is possible to "combine" the samples from A and C into a sample from B. The new rule is a generalization of our above ideas. We assign the sample from A to be a sample from B if the A's sample belongs to B (for technical reasons, we decrease the probability of this event by $\frac{|A|}{|B|}$ multiplicative factor). Otherwise, we assign the sample from C to be the sample from B. We show that this rule gives a uniform sample from B.

To apply this idea to sliding windows, we need to overcome two problems. First, we must be able to maintain such an A and C (as before we associate B with our window). This task is nontrivial, since the size of the window is unknown. Our second key idea is a novel *covering decomposition* structure. Using this structure, we are able to maintain such an A and C at any moment.

Second, we need the ability to generate events w.p. $\frac{|A|}{|B|}$ which is still an unknown probability since $|B|$ is the size of our window. Our third key idea is a novel technique that we call *generating implicit events*. At the heart of our technique lies the idea of gradually decreasing the probabilities, starting from 1, until we achieve the desired probability of $|A|/|B|$. In particular, we show that it is possible to generate a non-uniform distribution over the elements of A, where the probability of picking an element is a function of the element's timestamp (or index). The function is constructed in such a way that the probability of picking an element among the last i elements of A is equal to $\frac{i}{|C|+i}$. That is, the probability of picking an expired element is $\frac{|C|}{|B|}$. Since $|A| \leqslant |C|$ and since we know the values of $|A|$ and $|C|$, it is possible to generate events w.p. $\frac{|A|}{|B|}$. The details can be found in the main body of this paper.

1.3.3. Black-box reduction from sampling without replacement to sampling with replacement

Finally, we show that a k-sample without replacement may be generated from k independent samples, $R_0, \ldots, R_{k-1}$. We apply our fourth key idea, a *black-box reduction* from sampling without replacement to sampling with replacement. The novelty of our approach is based on sampling from different domains; in fact, R_i samples all but i last active elements. Such samples can be generated if, in addition, we store the last k elements.

1.3.4. Independence of disjoint windows

Our algorithms generate independent samples for non-overlapping windows. The independence follows from the nice property of the reservoir algorithm (that we use to generate samples in the buckets). Let R_1 be a sample generated for the

bucket $\tilde{B}$, upon arrival of i elements of $\tilde{B}$. Let R_2 be a fraction of the final sample (i.e., the sample when the last element of $\tilde{B}$ arrives) that belongs to the last $|\tilde{B}| - i$ elements. The reservoir algorithm implies that R_1 and R_2 are independent. Since the rest of the buckets contain independent samples as well, we conclude that our algorithms are independent for non-overlapping windows.

1.4. Roadmap and notations

We use the following notations throughout our paper. We denote by D a stream and by p_i, $i \geqslant 0$ its i-th element. For $0 \leqslant x < y$ we define $[x, y] = \{i,\ x \leqslant i \leqslant y\}$. Finally, *bucket* $B(x, y)$ is the set of all stream elements between p_x and p_{y-1}: $B(x, y) = \{p_i,\ i \in [x, y-1]\}$. We use $\log x$ notation for the logarithm of x to the base 2.

Our bounds are expressed in memory words; that is we assume that a single memory word is sufficient to store a stream element or its index or a timestamp.

Section 2 presents sampling for sequence-based windows, with and without replacement. Sections 3 and 4 are devoted to sampling for timestamp-based windows, with and without replacement. Section 5 outlines possible applications for our approach.

2. Equivalent-width partitions and sampling for sequence-based windows

2.1. Sampling with replacement

Let n be the predefined size of a window. We say that a bucket is *active* if all its elements have arrived and at least one element is non-expired. We say that a bucket is *partial* if not all of its elements have arrived. We show below how to create a single random sample. To create a k-random sample, we repeat the procedure k times, independently.

We divide D into buckets $B(in, (i+1)n)$, $i = 0, 1, \ldots$. At any point in time, we have exactly one active bucket and at most one partial bucket. For every such bucket B, we independently generate a single sample, using the reservoir algorithm [60]. We denote this sample by X_B.

Let B be a partial bucket and $C \subseteq B$ be the set of all arrived elements of B. The properties of the reservoir algorithm[2] imply that X_B is a random sample of C.

Below, we construct a random sample Z of all non-expired elements. Let U be the active bucket. If there is no partial bucket, then U contains only all non-expired elements. Therefore, $Z = X_U$ is a valid sample. Otherwise, let V be the partial bucket. Let $U_e = \{x\colon\ x \in U,\ x \text{ is expired}\}$, $U_a = \{x\colon\ x \in U,\ x \text{ is non-expired}\}$, $V_a = \{x\colon\ x \in V,\ x \text{ arrived}\}$.

Note that $|V_a| = |U_e|$ and let $s = |V_a|$. Also, note that our window is $U_a \cup V_a$ and X_V is a random sample of V_a. The random sample Z is constructed as follows. If X_U is not expired, we put $Z = X_U$, otherwise $Z = X_V$. To prove the correctness, let p be a non-expired element. If $p \in U_a$, then $P(Z = p) = P(X_U = p) = \frac{1}{n}$. If $p \in V_a$, then

$$P(Z = p) = P(X_U \in U_e, X_V = p) = P(X_U \in U_e)P(X_V = p) = \frac{s}{n}\frac{1}{s} = \frac{1}{n}.$$

Therefore, Z is a valid random sample. We need to store only samples of active or partial buckets. Since the number of such buckets is at most two and the reservoir algorithm requires $\Theta(1)$ memory, the total memory of our algorithm for k-sample is $\Theta(k)$. Thus,

Theorem 2.1. *It is possible to maintain k samples with replacement for sequence-based windows using $O(k)$ memory words.*

2.2. Sampling without replacement

We can generalize the idea above to provide k samples without replacement. In this section k-sample means k-random sampling without replacement.

We use the same buckets $B(in, (i+1)n)$, $i = 0, 1, \ldots$. For every such bucket B, we independently generate a k-sample X_B, using the reservoir algorithm.

Let B be a partial bucket and $C \subseteq B$ be the set of all arrived elements. The properties of the reservoir algorithm imply that either $X_B = C$, if $|C| < k$, or X_B is a k-sample of C. In both cases, we can generate an i-sample of C using X_B only, for any $0 < i \leqslant \min(k, |C|)$.

Our algorithm is as follows. Let U be the active bucket. If there is no partial bucket, then U contains only all active elements. Therefore, we can put $Z = X_U$. Otherwise, let V be the partial bucket. We define U_e, U_a, V_a, s as before and construct Z as follows. If all elements of X_U are not expired, $Z = X_U$. Otherwise, let i be the number of expired elements, $i = |U_e \cap X_U|$. As we mentioned before, we can generate an i-sample of V_a from X_V, since $i \leqslant \min(k, s)$. We denote this sample as X_V^i and put

$$Z = (X_U \cap U_a) \cup X_V^i.$$

[2] We recall that the reservoir algorithm [60] maintains a sample from an unbounded stream.

We will prove now that Z is a valid random sample. Let $Q = \{p_{j_1}, \ldots, p_{j_k}\}$ be a fixed set of k non-expired elements such that $j_1 < j_2 < \cdots < j_k$. Let $i = |Q \cap V_A|$, so $\{p_{j_1}, \ldots, p_{j_{k-i}}\} \subseteq U_a$ and $\{p_{j_{k-i+1}}, \ldots, p_{j_k}\} \subseteq V_a$. If $i = 0$, then $Q \subseteq U$ and

$$P(Z = Q) = P(X_U = Q) = \frac{1}{\binom{n}{k}}.$$

Otherwise, by independence of X_U and X_V^i

$$\begin{aligned} P(Z = Q) &= P\big(|X_U \cap U_e| = i,\ \{p_{j_1}, \ldots, p_{j_{k-i}}\} \subseteq X_U,\ X_V^i = \{p_{j_{k-i+1}}, \ldots, p_{j_k}\}\big) \\ &= P\big(|X_U \cap U_e| = i,\ \{p_{j_1}, \ldots, p_{j_{k-i}}\} \subseteq X_U\big) * P\big(X_V^i = \{p_{j_{k-i+1}}, \ldots, p_{j_k}\}\big) \\ &= \frac{\binom{s}{i}}{\binom{n}{k}} * \frac{1}{\binom{s}{i}} = \frac{1}{\binom{n}{k}}. \end{aligned}$$

Therefore, Z is a valid random sample of non-expired elements. Note that we store only samples of active or partial buckets. Since the number of such buckets is at most two and the reservoir algorithm requires $O(k)$ memory, the total memory of our algorithm is $O(k)$. Thus,

Theorem 2.2. *It is possible to maintain k samples without replacement for sequence-based windows using $O(k)$ memory words.*

3. Sampling with replacement for timestamp-based windows

Let $n = n(t)$ be the number of non-expired elements. For each element p, the timestamp $T(p)$ represents the moment of p's entrance. For a window with (predefined) parameter t_0, p is active at time t if $t - T(p) < t_0$. We show below how to create a single random sample. To create a k-random sample, we repeat the procedure k times, independently.

3.1. Notations

A bucket structure $BS(x, y)$ is a tuple

$$\big\{p_x, x, y, T(p_x), R_{x,y}, Q_{x,y}, r, q\big\},$$

where $T(p_x)$ is the timestamp of p_x, $R_{x,y}$ and $Q_{x,y}$ are independent random samples from $B(x, y)$ and r, q are indexes of the picked (for random samples) elements. We denote by $N(t)$ the index of the last element of D at the moment t and by $l(t)$ the index of the earliest active element at the moment t. Note that $N(t) \leqslant N(t+1)$, $l(t) \leqslant l(t+1)$ and $T(p_i) \leqslant T(p_{i+1})$.

3.2. Covering decomposition

Definition 3.1. Let $a \leqslant b$ be two indexes. A covering decomposition of a bucket $B(a, b)$, $\zeta(a, b)$, is an ordered set of bucket structures with independent samples inductively defined below

$$\zeta(b, b) := BS(b, b+1),$$

and for $a < b$,

$$\zeta(a, b) := \big\langle BS(a, c), \zeta(c, b)\big\rangle,$$

where $c = a + 2^{\lfloor \log(b+1-a)\rfloor - 1}$.

Note that

$$\big|\zeta(a, b)\big| = O\big(\log(b - a)\big),$$

so $\zeta(a, b)$ uses $O(\log(b - a))$ memory.

Given p_{b+1}, we inductively define an operator $Incr(\zeta(a, b))$ as follows.

$$Incr\big(\zeta(b, b)\big) := \big\langle BS(b, b+1), BS(b+1, b+2)\big\rangle.$$

For $a < b$, we put

$$Incr\big(\zeta(a, b)\big) := \big\langle BS(a, v), Incr\big(\zeta(v, b)\big)\big\rangle,$$

where v is defined below.

If $\lfloor \log(b+2-a)\rfloor = \lfloor \log(b+1-a)\rfloor$, then we put $v = c$, where $BS(a, c)$ is the first bucket structure of $\zeta(a, b)$. Otherwise, we put $v = d$, where $BS(c, d)$ is the second bucket structure of $\zeta(a, b)$. (Note that $\zeta(a, b)$ contains at least two buckets for $a < b$.)

We show how to construct $BS(a,d)$ from $BS(a,c)$ and $BS(c,d)$. We have in this case $\lfloor\log(b+2-a)\rfloor = \lfloor\log(b+1-a)\rfloor+1$, and therefore $b+1-a = 2^i-1$ for some $i \geqslant 2$. Thus $c-a = 2^{\lfloor\log(2^i-1)\rfloor-1} = 2^{i-2}$ and

$$\lfloor\log(b+1-c)\rfloor = \lfloor\log\bigl(b+1-a-(c-a)\bigr)\rfloor = \lfloor\log\bigl(2^i-2^{i-2}-1\bigr)\rfloor = i-1.$$

Thus $d-c = 2^{\lfloor\log(b+1-c)\rfloor-1} = 2^{i-2} = c-a$. Now we can create $BS(a,v)$ by unifying $BS(a,c)$ and $BS(c,d)$: $BS(a,v) = \{p_a, a, v, T(p_a), R_{a,d}, Q_{a,d}, r', q'\}$. We put $R_{a,d} = R_{a,c}$ with probability $\frac{1}{2}$ and $R_{a,d} = R_{c,d}$ otherwise. Since $d-c = c-a$, and $R_{c,d}, R_{a,c}$ are distributed uniformly, we conclude that $R_{a,d}$ is distributed uniformly as well. $Q_{a,d}$ is defined similarly and r', q' are indexes of the chosen samples. Finally, the new samples are independent of the rest of ζ's samples. Note also that $Incr(\zeta(a,b))$ requires $O(\log(b-a))$ operations.

Thus, we conclude

Fact 3.2. The size of covering decomposition $\zeta(a,b)$ is bounded by $O(\log(b-a))$. $Incr(\zeta(a,b))$ requires $O(\log(b-a))$ time and memory.

Definition 3.3. Two bucket structures $BS(a,b)$ and $BS(c,d)$ are equal if $a=b$ and $c=d$. Two sequences of bucket structures are equal if the sequences are of the same size and their i-th elements are equal for each i.

Lemma 3.4. *For any a and b, the lists $Incr(\zeta(a,b))$ and $\zeta(a,b+1)$ are equal (in the sense of Definition 3.3).*

Proof. We prove the lemma by induction on $b-a$. If $a=b$ then, since $b+1 = b+2^{\lfloor\log((b+1)+1-b)\rfloor-1}$, we have, by definition of $\zeta(b,b+1)$,

$$\zeta(b,b+1) = \bigl\langle BS(b,b+1), \zeta(b+1,b+1)\bigr\rangle = \bigl\langle BS(b,b+1), BS(b+1,b+2)\bigr\rangle = Incr\bigl(\zeta(b,b)\bigr).$$

We assume that the lemma is correct for $b-a<h$ and prove it for $b-a=h$. Let $BS(a,v)$ be the first bucket of $Incr(\zeta(a,b))$. Let $BS(a,c)$ be the first bucket of $\zeta(a,b)$. By definition, if $\lfloor\log(b+2-a)\rfloor = \lfloor\log(b+1-a)\rfloor$ then $v=c$. We have

$$v = a+2^{\lfloor\log(b+1-a)\rfloor-1} = a+2^{\lfloor\log(b+2-a)\rfloor-1}.$$

Otherwise, let $BS(c,d)$ be the second bucket of $\zeta(a,b)$. We have from the above $\lfloor\log(b+2-a)\rfloor = \lfloor\log(b+1-a)\rfloor+1$, $d-c=c-a$ and $v=d$. Thus

$$v = d = 2c-a = 2\bigl(a+2^{\lfloor\log(b+1-a)\rfloor-1}\bigr)-a = a+2^{\lfloor\log(b+1-a)\rfloor} = a+2^{\lfloor\log(b+2-a)\rfloor-1}.$$

In both cases $v = a+2^{\lfloor\log((b+1)+1-a)\rfloor-1}$ and, by definition of ζ

$$\zeta(a,b+1) = \bigl\langle BS(a,v), \zeta(v,b+1)\bigr\rangle.$$

By induction, since $b-v<h$, we have $Incr(\zeta(v,b)) = \zeta(v,b+1)$. Thus

$$\zeta(a,b+1) = \bigl\langle BS(a,v), \zeta(v,b+1)\bigr\rangle = \bigl\langle BS(a,v), Incr\bigl(\zeta(v,b)\bigr)\bigr\rangle = Incr\bigl(\zeta(a,b)\bigr). \qquad \square$$

Lemma 3.5. *For any t with a positive number of active elements, we are able to maintain one of the following*:

1. *$\zeta(l(t),N(t))$, or*
2. *$BS(y_t,z_t), \zeta(z_t,N(t))$, where $y_t < l(t) \leqslant z_t$, $z_t - y_t \leqslant N(t)+1-z_t$ and all random samples are independent. Furthermore, the memory and time required are $O(\log(n(t)))$.*

Proof. We prove the lemma by induction on t. First we assume that $t=0$. If no element arrives at time 0, the stream is empty and we do nothing. Otherwise, we put $\zeta(0,0) = BS(0,1)$, and for any i, $0<i\leqslant N(0)$ we generate $\zeta(0,i)$ by executing $Incr(\zeta(0,i-1))$. At the end of this step, we have $\zeta(0,N(0)) = \zeta(l(0),N(0))$; thus, the first condition of the lemma is valid.

We assume that the lemma is correct for t and prove it for $t+1$.

1. If for t the window is empty, then the procedure is the same as for $t=0$.
2. If for t the first condition of the lemma is true then we have three sub-cases.
   (a) If $p_{l(t)}$ is not expired at the moment $t+1$, then $l(t+1) = l(t)$. Similar to the basic case, we apply the *Incr* procedure for every new element with index i, $N(t) < i \leqslant N(t+1)$. Due to the properties of *Incr*, we have at the end $\zeta(l(t+1), N(t+1))$. Therefore the first condition of the lemma is true for $t+1$.
   (b) If $p_{N(t)}$ is expired, then our current bucket structures represent only expired elements. We delete them and apply the procedure for the basic case.
   (c) The last sub-case is the one when $p_{N(t)}$ is not expired and $p_{l(t)}$ is expired. Let $\langle BS_1,\ldots,BS_h\rangle$, $(BS_i = BS(v_i,v_{i+1}))$ be all buckets of $\zeta(l(t),N(t))$. Since $p_{N(t)}$ is not expired, there exists exactly one bucket structure, BS_i, such that p_{v_i} is

expired and $p_{v_{i+1}}$ is not expired. We can find it by checking all the bucket structures, since we store timestamps. We put

$$y_{t+1} = v_i, \qquad z_{t+1} = v_{i+1}.$$

We have by definition

$$\zeta\big(z_{t+1}, N(t)\big) = \zeta\big(v_{i+1}, N(t)\big) = \langle BS_{i+1}, \ldots, BS_k\rangle.$$

Applying the *Incr* procedure to all new elements, we construct $\zeta(z_{t+1}, N(t+1))$. Finally, we have:

$$z_{t+1} - y_{t+1} = v_{i+1} - v_i = 2^{\lfloor \log(N(t)+1-v_i)\rfloor - 1} \leqslant \frac{1}{2}\big(N(t)+1-v_i\big) = \frac{1}{2}\big(N(t)+1-y_{t+1}\big).$$

Therefore, and by Definition 3.1, we have $z_{t+1} - y_{t+1} \leqslant N(t)+1-z_{t+1} \leqslant N(t+1)+1-z_{t+1}$. Thus, the second condition of the lemma is valid for $t+1$. We discard all non-used bucket structures $BS_1, \ldots, BS_{i-1}$.

3. Finally, assume that for t we maintain case of the second condition of the lemma. Similarly, we have three sub-cases.
   (a) If p_{z_t} is not expired at the moment $t+1$, we put $y_{t+1} = y_t$, $z_{t+1} = z_t$. We have

   $$z_{t+1} - y_{t+1} = z_t - y_t \leqslant N(t)+1-z_t \leqslant N(t+1)+1-z_{t+1}.$$

   Again, we add the new elements using the *Incr* procedure and we construct $\zeta(z_{t+1}, N(t+1))$. Therefore the second condition of the lemma is true for $t+1$.
   (b) If $p_{N(t)}$ is expired, we apply exactly the same procedure as for 2(b).
   (c) If p_{z_t} is expired and $p_{N(t)}$ is not expired, we apply exactly the same procedure as for 2(c).

Note also that $y_t \leqslant l(t)$ and that $n(t) \leqslant N(t) - l(t)$. Thus, memory and time bounds follow from Fact 3.2. Therefore, the lemma is correct. □

3.3. Generating implicit events

We use the following notations for this section. Let $B_1 = B(a,b)$ and $B_2 = B(b, N(t)+1)$ be two buckets such that p_a is expired, p_b is active and $|B_1| \leqslant |B_2|$. Let BS_1 and BS_2 be corresponding bucket structures, with independent random samples R_1, Q_1 and R_2, Q_2. We put $\alpha = b - a$ and $\beta = N(t) + 1 - b$. Let γ be the (unknown) number of non-expired elements inside B_1, so $n = \beta + \gamma$. We stress that α, β are known and γ is unknown.

Lemma 3.6. *It is possible to generate a random sample $Y = Y(Q_1)$ of B_1, with the following distribution*:

$$P(Y = p_{b-i}) = \frac{\beta}{(\beta+i)(\beta+i-1)}, \quad 0 < i < \alpha,$$
$$P(Y = p_a) = \frac{\beta}{\beta+\alpha-1}.$$

Y is independent of R_1, R_2, Q_2 and can be generated within constant memory and time, using Q_1.

Proof. Let $\{H_j\}_{j=1}^{\alpha-1}$ be a set of zero-one independent random variables such that

$$P(H_j = 1) = \frac{\alpha\beta}{(\beta+j)(\beta+j-1)}.$$

Let $D = B_1 \times \{0,1\}^{\alpha-1}$ and Z be the random vector with values from D, $Z = \langle Q_1, H_1, \ldots, H_{\alpha-1}\rangle$. Let $\{A_i\}_{i=1}^{\alpha}$ be a set of subsets of D:

$$A_i = \big\{\langle q_{b-i}, a_1, \ldots, a_{i-1}, 1, a_{i+1}, \ldots, a_{\alpha-1}\rangle \;\big|\; a_j \in \{0,1\},\ j \neq i\big\}.$$

Finally we define Y as follows

$$Y = \begin{cases} q_{b-i}, & \text{if } Z \in A_i,\ 1 \leqslant i < \alpha, \\ q_a, & \text{otherwise.} \end{cases}$$

Since Q_1 is independent of R_1, R_2, Q_2, Y is independent of them as well. We have

$$\begin{aligned} P(Y = p_{b-i}) = P(Z \in A_i) &= P\big(Q_1 = q_{b-i},\ H_i = 1,\ H_j \in \{0,1\} \text{ for } j \neq i\big) \\ &= P(Q_1 = q_{b-i})P(H_i = 1)P\big(H_j \in \{0,1\} \text{ for } j \neq i\big) \\ &= P(Q_1 = q_{b-i})P(H_i = 1) = \frac{1}{\alpha}\frac{\alpha\beta}{(\beta+i)(\beta+i-1)} = \frac{\beta}{(\beta+i)(\beta+i-1)}. \end{aligned}$$

Also,

$$P(Y=p_a)=1-\sum_{i=1}^{\alpha-1}P(Y=p_{b-i})=1-\sum_{i=1}^{\alpha-1}\frac{\beta}{(\beta+i)(\beta+i-1)}$$
$$=1-\beta\sum_{i=1}^{\alpha-1}\left(\frac{1}{\beta+i-1}-\frac{1}{\beta+i}\right)=1-\beta(\frac{1}{\beta}-\frac{1}{\beta+\alpha-1})=\frac{\beta}{\beta+\alpha-1}.$$

By definition of A_i, the value of Y is uniquely defined by Q_1 and exactly one H. Therefore, the generation of the whole vector Z is not necessary. Instead, we can calculate Y by the following simple procedure. Once we know the index of Q_1's value, we generate the corresponding H_i and calculate the value of Y. We can omit the generation of other Hs, and therefore we need constant time and memory. □

Lemma 3.7. *It is possible to generate a zero-one random variable X such that $P(X=1)=\frac{\alpha}{\beta+\gamma}$. X is independent of R_1, R_2, Q_2 and can be generated using constant time and memory.*

Proof. Since γ is unknown, it cannot be generated by flipping a coin; a slightly more complicated procedure is required.

Let $Y(Q_1)$ be the random variable from Lemma 3.6. We have

$$P(Y \text{ is not expired})=\sum_{i=1}^{\gamma}P(Y=q_{b-i})=\sum_{i=1}^{\gamma}\frac{\beta}{(\beta+i)(\beta+i-1)}$$
$$=\beta\sum_{i=1}^{\gamma}\left(\frac{1}{\beta+i-1}-\frac{1}{\beta+i}\right)=\beta\left(\frac{1}{\beta}-\frac{1}{\beta+\gamma}\right)=\frac{\gamma}{\beta+\gamma}.$$

Therefore $P(\text{Y is expired})=\frac{\beta}{\beta+\gamma}$.

Let S be a zero-one variable, independent of R_1, R_2, Q_2, Y such that

$$P(S=1)=\frac{\alpha}{\beta}.$$

We put

$$X=\begin{cases}1, & \text{if } Y \text{is expired AND } S=1,\\ 0, & \text{otherwise.}\end{cases}$$

We have

$$P(X=1)=P(Y \text{ is expired},\ S=1)=P(Y \text{ is expired})P(S=1)=\frac{\beta}{\beta+\gamma}\frac{\alpha}{\beta}=\frac{\alpha}{\beta+\gamma}.$$

Since Y and S are independent of R_1, R_2, Q_2, X is independent of them as well. Since we can determine if Y is expired within constant time, we need a constant amount of time and memory. □

Lemma 3.8. *It is possible to construct a random sample V of all non-expired elements using only the data of BS_1, BS_2 and constant time and memory.*

Proof. Our goal is to generate a random variable V that chooses a non-expired element w.p. $\frac{1}{\beta+\gamma}$. Let X be the random variable generated in Lemma 3.7. We define V as follows.

$$V=\begin{cases}R_1, & R_1 \text{ is not expired AND } X=1,\\ R_2, & \text{otherwise.}\end{cases}$$

Let p be a non-expired element. If $p\in B_1$, then since X is independent of R_1, we have

$$P(V=p)=P(R_1=p, X=1)=P(R_1=p)P(X=1)=\frac{1}{\alpha}\frac{\alpha}{\beta+\gamma}=\frac{1}{\beta+\gamma}=\frac{1}{n}.$$

If $p\in B_2$, then

$$P(V=p)=\big(1-P(R_1 \text{ is not expired})P(X=1)\big)P(R_2=p)=\left(1-\frac{\gamma}{\alpha}\frac{\alpha}{\beta+\gamma}\right)\frac{1}{\beta}=\frac{1}{\beta+\gamma}=\frac{1}{n}.$$ □

3.4. Main results

Theorem 3.9. *We can maintain a random sample over all non-expired elements using $\Theta(\log n)$ memory.*

Proof. By using Lemma 3.5, we are able to maintain one of two cases. If case 1 occurs, we can combine random variables of all bucket structures with appropriate probabilities and get a random sample of all non-expired elements. If case 2 occurs, we use notations of Section 3.3, interpret the first bucket as B_1 and combine buckets of the covering decomposition to generate samples from B_2. Properties of the second case imply $|B_1| \leqslant |B_2|$ and therefore, by using Lemma 3.8, we are able to produce a random sample as well. All procedures described in the lemmas require $\Theta(\log n)$ memory. Therefore, the theorem is correct. □

Lemma 3.10. *The memory usage of maintaining a random sample within a timestamp-based window has a lower bound $\Omega(\log(n))$.*

Proof. Let D be a stream with the following property. For timestamp i, $0 \leqslant i \leqslant 2t_0$, we have 2^{2t_0-i} elements and for $i > 2t_0$, we have exactly one element per timestamp.

For timestamp $0 \leqslant i \leqslant t_0$, the probability of choosing p with $T(p) = i$ at the moment $t_0 + i - 1$ is

$$\frac{2^{2t_0-i}}{\sum_{j=i}^{i+t_0-1} 2^{2t_0-j}} = \frac{2^{2t_0-i}}{2^{t_0-i+1}\sum_{j=0}^{t_0-1} 2^{t_0-j-1}} = \frac{2^{t_0-1}}{\sum_{j=0}^{t_0-1} 2^j} = \frac{2^{t_0-1}}{2^{t_0}-1} > \frac{1}{2}.$$

Therefore, the expected number of distinct timestamps that will be picked between moments $t_0 - 1$ and $2t_0 - 1$ is at least $\sum_{i=t_0-1}^{2t_0-1} \frac{1}{2} = \frac{t_0+1}{2}$. So, with a positive probability we need to keep in memory at least $\frac{t_0}{2}$ distinct elements at the moment t_0. The number of active elements n at this moment is at least 2^{t_0}. Therefore the memory usage at this moment is $\Omega(\log n)$, with positive probability. □

4. Black-box reduction

In this section, we present black-box reduction from sampling without replacement to sampling with replacement. As a result, we obtain an optimal algorithm for sampling without replacement for timestamp-based windows. Informally, the idea is as follows. We maintain k independent random samples $R_0, \ldots, R_{k-1}$ of active elements, using the algorithm from Section 3. The difference between these samples and the k-sample with replacement is that R_i samples all active elements except the last i. This can be done using $O(k + k \log n)$ memory. Finally, a k-sample without replacement can be generated using $R_0, \ldots, R_{k-1}$ only.

Let us describe the algorithm in detail. First, we construct R_i. To do this, we maintain an auxiliary array with the last i elements. We repeat all procedures in Section 3, but we "delay" the last i elements. An element is added to covering decomposition only when more than i elements arrive after it. We prove the following variant of Lemma 3.5.

Lemma 4.1. *Let $0 < i \leqslant k$. For any t with more than i active elements, we are able to maintain one of the following*:

1. *$\zeta(l(t), N(t) - i)$, or*
2. *$BS(y_t, z_t), \zeta(z_t, N(t) - i)$, where $y_t < l(t) \leqslant z_t$ and $z_t - y_t \leqslant N(t) + 1 - i - z_t$ and all random samples of the bucket structures are independent. Furthermore, the memory and time required are $O(\log(n(t)))$.*

Proof. The proof is the same as in Lemma 3.5, except for cases 1, 2(b), 3(b). For these cases, when the current window is empty, we keep it empty unless more than i elements are active. We can do this using our auxiliary array. Also, when new elements arrive, some of them may already be expired (if we kept them in the array). We therefore cannot apply the *Incr* procedure for any "new" element. Instead, we should first skip all expired elements and then apply *Incr*. The rest of the proof remains the same. □

The rest of the procedure remains the same. Note that we can use the same array for every i, and therefore we can construct $R_0, \ldots, R_{k-1}$ using $\Theta(k + k \log n)$ memory.

In the reminder of this section, we show how $R_0, \ldots, R_{k-1}$ can be used to generate a k-sample without replacement. We denote by S_i^j an i-random sample without replacement from $[1, j]$.

Lemma 4.2. *S_{a+1}^{b+1} can be generated using independent S_a^b, S_1^{b+1} samples only.*

Proof. The algorithm is as follows.

$$S_{a+1}^{b+1} = \begin{cases} S_a^b \cup \{b+1\}, & \text{if } S_1^{b+1} \in S_a^b, \\ S_a^b \cup S_1^{b+1}, & \text{otherwise.} \end{cases}$$

Let $X = \{x_1, \ldots, x_{a+1}\}$ be a set of points from $[1, b+1]$, such that $x_1 < x_2 < \cdots < x_a < x_{a+1}$.
If $x_{a+1} < b+1$, then we have

$$P\left(S_{a+1}^{b+1} = X\right) = P\left(\bigcup_{j=1}^{a+1}\left(S_1^{b+1} = x_j \wedge S_a^b = X\backslash\{x_j\}\right)\right)$$
$$= \sum_{j=1}^{a+1} P\left(S_1^{b+1} = x_j\right) P\left(S_a^b = X\backslash\{x_j\}\right) = (a+1)\frac{1}{b+1}\frac{1}{\binom{b}{a}} = \frac{1}{\binom{b+1}{a+1}}.$$

Otherwise,

$$P\left(S_{a+1}^{b+1} = X\right) = P\left(S_a^b = X\backslash\{b+1\}, S_1^{b+1} \in X\right) = \frac{1}{\binom{b}{a}}\frac{a+1}{b+1} = \frac{1}{\binom{b+1}{a+1}}. \qquad \square$$

Lemma 4.3. *S_k^n can be generated using only independent samples $S_1^n, S_1^{n-1}, \ldots, S_1^{n-k+1}$.*

Proof. By using Lemma 4.2, we can generate S_2^{n-k+2} using S_1^{n-k+1} and S_1^{n-k+2}. We can repeat this procedure and generate S_j^{n-k+j}, $2 \leqslant j \leqslant k$, using $S_{j-1}^{n-k+j-1}$ (that we already constructed by induction) and S_1^{n-k+j}. For $j = k$ we have S_k^n. $\square$

By using Lemma 4.3, we can generate a k-sample without replacement using only $R_0, \ldots, R_{k-1}$. Thus, we have proved

Theorem 4.4. *It is possible to maintain k samples without replacement for timestamp-based windows using $O(k \log n)$ memory words.*

5. Applications

Consider an algorithm Λ that is sampling-based, i.e., it operates on a uniformly chosen subset of D instead of the whole stream. Such an algorithm can be immediately transformed to sliding windows by replacing the underlying sampling method with our algorithms. We obtain the following general result and illustrate it with the examples below.

Theorem 5.1. *For a sampling-based algorithm Λ that solves problem P, there exists an algorithm Λ' that solves P on sliding windows. The memory guarantees are preserved for sequence-based windows and have a multiplicative overhead of $\log n$ for timestamp-based windows.*

Frequency moment estimation is a fundamental problem in data stream processing. Given a stream of elements, such that $p_j \in [m]$, the frequency x_i of each $i \in [m]$ is defined as $|\{j \mid p_j = i\}|$ and the k-th frequency moment is defined as $F_k = \sum_{i=1}^m x_i^k$. The first algorithm for frequency moments for $k > 2$ was proposed in the seminal paper of Alon, Matias and Szegedy [4]. They present an algorithm that uses $O(m^{1-\frac{1}{k}})$ memory. Numerous improvements to lower and upper bounds have been reported, including the works of Bar-Yossef, Jayram, Kumar and Sivakumar [14], Chakrabarti, Khot and Sun [22], Coppersmith and Kumar [28], and Ganguly [37]. Finally, Indyk and Woodruff [49] and later Bhuvanagiri, Ganguly, Kesh and Saha [17] presented algorithms that use $\tilde{O}(m^{1-\frac{2}{k}})$ memory and are optimal. The algorithm of Alon, Matias and Szegedy [4] is sampling-based, thus we can adapt it to sliding windows using our methods.

Corollary 5.2.[3] *For any $k > 2$ and for any $\epsilon, \delta > 0$ there exists an algorithm that maintains a $(1 \pm \epsilon, \delta)$-approximation of the k-th frequency moment over sliding windows using $\tilde{O}(m^{1-\frac{1}{k}})$ bits.*[4]

Recently, numerous graph problems were addressed in the streaming environment. Stream elements represent edges of the graph, given in arbitrary order (we refer the readers to [19] for a detailed explanation of the model). One of the fundamental graph problems is estimating a number of small cliques in a graph, in particular the number of triangles. Effective solutions were proposed by Jowhari and Ghodsi [51], Bar-Yossef, Kumar and Sivakumar [15] and Buriol, Frahling, Leonardi, Marchetti-Spaccamela and Sohler [19]. The last paper presented an (ϵ, δ)-approximation algorithm that uses $O(1 + \frac{\log|E|}{|E|}\frac{1}{\epsilon^2}\frac{|T_1|+2|T_2|+3|T_3|}{|T_3|}\log\frac{2}{\delta})$ memory [19, Theorem 2] that is the best result so far. Here, $|T_i|$ represents the number of node-triplets having i edges in the induced sub-graph. The algorithm is applied on a random sample collected using the reservoir method. By replacing the reservoir sampling with our algorithms, we obtain the following result.

[3] For sufficiently large (yet polylogarithmic) values of k we improve the result of Braverman and Ostrovsky [18] that uses $\tilde{O}(k^k m^{1-\frac{2}{k}})$ space. However, for constant k our result is not optimal and [18] gives an optimal memory bound. Thus these two results are incomparable.

[4] An algorithm maintains $(1 \pm \epsilon, \delta)$-approximation of function f if at any moment the algorithm outputs f' s.t. $(1-\epsilon)f(D) \leqslant f'(D) \leqslant (1+\epsilon)f(D)$ w.p. at least $1-\delta$. We denote $\tilde{O}(f(m)) = \frac{1}{\epsilon^{O(1)}}(\log m)^{O(1)}(\log n)^{O(1)}f(m)$.

Corollary 5.3. *There exists an algorithm that maintains an (ϵ, δ)-approximation of the number of triangles over sliding windows. For sequence-based windows it uses $O(1 + \frac{\log|E_W|}{|E_W|}\frac{1}{\epsilon^2}\frac{|T_1|+2|T_2|+3|T_3|}{|T_3|}\log\frac{2}{\delta})$ memory, where E_W is the set of active edges. Timestamp-based windows adds a multiplicative factor of* $\log n$.

Following [19], our method is also applicable for incidence streams, where all edges of the same vertex come together.

The entropy of a stream is defined as $H = -\sum_{i=1}^{m}\frac{x_i}{N}\log\frac{x_i}{N}$, where x_i is a frequency and N is a length of the stream. The entropy norm is defined as $F_H = \sum_{i=1}^{m} x_i \log x_i$. Effective solutions for entropy and entropy norm estimations were recently reported by Guha, McGregor and Venkatasubramanian [45]; Chakrabarti, Do Ba and Muthukrishnan [21]; Harvey, Nelson and Onak [48]; Chakrabarti, Cormode and McGregor [20]; and Zhao, Lall, Ogihara, Spatscheck, Wang and Xu [62].

The paper of Chakrabarti, Cormode and McGregor presents an algorithm that is based on a variation of reservoir sampling. The algorithm maintains entropy using $O(\epsilon^{-2}\log\delta^{-1}\log(N))$ memory bits that is nearly optimal. The authors also considered the sliding window model and used a variant of priority sampling [10] to obtain the approximation. Thus, the worst-case memory guarantees are not preserved for sliding windows. By replacing priority sampling with our methods, we obtain

Corollary 5.4. *There exists an algorithm that maintains an (ϵ, δ)-approximation of entropy on sliding windows using $O(\epsilon^{-2}\log\delta^{-1}\log n\log(N))$ memory bits for timestamp-based windows and $O(\epsilon^{-2}\log\delta^{-1}\log(N))$ memory bits for sequence-based windows.*

Moreover, our methods can be used with the algorithm from [21] to obtain $\tilde{O}(1)$ memory for large values of the entropy norm. This algorithm is based on reservoir sampling and thus can be straightforwardly implemented in sliding windows. As a result, we build the first solutions with provable memory guarantees on sliding windows.

Our algorithms can be naturally extended to some biased functions. Biased sampling [2] is non-uniform, giving larger probabilities for more recent elements. The distribution is defined by a biased function. We can apply our methods to implement step biased functions, maintaining samples over each window with different lengths and combining the samples with corresponding probabilities. Our algorithm can extend the ideas of Feigenbaum, Kannan, Strauss and Viswanathan [35] for testing and spot-checking to sliding windows. Finally, we can apply our tools to the algorithm of Procopiuc and Procopiuc for density estimation [57], since it is based on the reservoir algorithm as well.

References

[1] C. Aggarwal (Ed.), Data Streams: Models and Algorithms, Springer-Verlag, 2007.
[2] C. Aggarwal, On biased reservoir sampling in the presence of stream evolution, in: Proceedings of the 32nd International Conference on Very Large Data Bases, 2006, pp. 607–618.
[3] N. Alon, N. Duffield, C. Lund, M. Thorup, Estimating arbitrary subset sums with few probes, in: Proceedings of the 24th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, 2005, pp. 317–325.
[4] N. Alon, Y. Matias, M. Szegedy, The space complexity of approximating the frequency moments, in: Proceedings of the 28th Annual ACM Symposium on Theory of Computing, 1996, pp. 20–29.
[5] A. Arasu, B. Babcock, S. Babu, J. Cieslewicz, M. Datar, K. Ito, R. Motwani, U. Srivastava, J. Widom, STREAM: The Stanford Data Stream Management System. Data-Stream Management: Processing High-Speed Data Streams, Springer-Verlag, 2005. Book chapter.
[6] A. Arasu, G.S. Manku, Approximate counts and quantiles over sliding windows, in: Proceedings of the 23rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, 2004.
[7] A.M. Ayad, J.F. Naughton, Static optimization of conjunctive queries with sliding windows over infinite streams, in: Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data, 2004.
[8] B. Babcock, S. Babu, M. Datar, R. Motwani, J. Widom, Models and issues in data stream systems, in: Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, 2002.
[9] B. Babcock, S. Babu, M. Datar, R. Motwani, D. Thomas, Operator scheduling in data stream systems, VLDB J. 13 (4) (2004) 333–353.
[10] B. Babcock, M. Datar, R. Motwani, Sampling from a moving window over streaming data, in: Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms, 2002, pp. 633–634.
[11] B. Babcock, M. Datar, R. Motwani, Load shedding for aggregation queries over data streams, in: Proceedings of the 20th International Conference on Data Engineering, 2004.
[12] B. Babcock, M. Datar, R. Motwani, L. O'Callaghan, Maintaining variance and k-medians over data stream windows, in: Proceedings of the 22nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, 2003, pp. 633–634.
[13] Z. Bar-Yossef, Sampling lower bounds via information theory, in: STOC, 2003.
[14] Z. Bar-Yossef, T.S. Jayram, R. Kumar, D. Sivakumar, An information statistics approach to data stream and communication complexity, in: Proceedings of the 43rd Symposium on Foundations of Computer Science, 2002, pp. 209–218.
[15] Z. Bar-Yossef, R. Kumar, D. Sivakumar, Reductions in streaming algorithms, with an application to counting triangles in graphs, in: Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms, 2002, pp. 623–632.
[16] Z. Bar-Yossef, R. Kumar, D. Sivakumar, Sampling algorithms: lower bounds and applications, in: STOC, 2001.
[17] L. Bhuvanagiri, S. Ganguly, D. Kesh, C. Saha, Simpler algorithm for estimating frequency moments of data streams, in: Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithm, 2006, pp. 708–713.
[18] V. Braverman, R. Ostrovsky, Smooth histograms on stream windows, in: Proceedings of the 48th Symposium on Foundations of Computer Science, 2007.
[19] L.S. Buriol, G. Frahling, S. Leonardi, A. Marchetti-Spaccamela, C. Sohler, Counting triangles in data streams, in: Proceedings of the 25th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, 2006, pp. 253–262.
[20] A. Chakrabarti, G. Cormode, A. McGregor, A near-optimal algorithm for computing the entropy of a stream, in: Proceedings of ACM-SIAM Symposium on Discrete Algorithms, 2007.

[21] A. Chakrabarti, K. Do Ba, S. Muthukrishnan, Estimating entropy and entropy norm on data streams, in: Proceedings of the 23rd International Symposium on Theoretical Aspects of Computer Science, 2006.
[22] A. Chakrabarti, S. Khot, X. Sun, Near-optimal lower bounds on the multi-party communication complexity of set-disjointness, in: Proceedings of the 18th Annual IEEE Conference on Computational Complexity, 2003.
[23] K. Chaudhuri, N. Mishra, When random sampling preserves privacy, in: CRYPTO, 2006.
[24] S. Chaudhuri, R. Motwani, V. Narasayya, On random sampling over joins, in: Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data, 1999, pp. 263–274.
[25] E. Cohen, Size-estimation framework with applications to transitive closure and reachability, J. Comput. System Sci. 55 (3) (1997) 441–453.
[26] E. Cohen, H. Kaplan, Summarizing data using bottom-k sketches, in: Proceedings of the 26th Annual ACM Symposium on Principles of Distributed Computing, 2007.
[27] G. Cormode, S. Muthukrishnan, I. Rozenbaum, Summarizing and mining inverse distributions on data streams via dynamic inverse sampling, in: Proceedings of the 31st International Conference on Very Large Data Bases, 2005.
[28] D. Coppersmith, R. Kumar, An improved data stream algorithm for frequency moments, in: Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms, 2004, pp. 151–156.
[29] A. Das, J. Gehrke, M. Riedewald, Semantic approximation of data stream joins, IEEE Trans. Knowl. Data Eng. 17 (1) (2005) 44–59.
[30] A. Dasgupta, P. Drineas, B. Harb, R. Kumar, M.W. Mahoney, Sampling algorithms and coresets for l_p regression, in: SODA, 2008.
[31] M. Datar, A. Gionis, P. Indyk, R. Motwani, Maintaining stream statistics over sliding windows: (extended abstract), in: Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms, 2002, pp. 635–644.
[32] M. Datar, S. Muthukrishnan, Estimating rarity and similarity over data stream windows, in: Proceedings of the 10th Annual European Symposium on Algorithms, 2002, pp. 323–334.
[33] N. Duffield, C. Lund, M. Thorup, Flow sampling under hard resource constraints, in: ACM SIGMETRICS Performance Evaluation Review, vol. 32, no. 1, 2004.
[34] J. Feigenbaum, S. Kannan, J. Zhang, Computing diameter in the streaming and sliding-window models, Algorithmica 41 (2005) 25–41.
[35] J. Feigenbaum, S. Kannan, M. Strauss, M. Viswanathan, Testing and spot-checking of data streams, Algorithmica 34 (1) (2002) 67–80.
[36] G. Frahling, P. Indyk, C. Sohler, Sampling in dynamic data streams and applications, in: Proceedings of the 21st Annual Symposium on Computational Geometry, 2005.
[37] S. Ganguly, Estimating frequency moments of update streams using random linear combinations, in: Proceedings of the 8th International Workshop on Randomized Algorithms, 2004, pp. 369–380.
[38] S. Gandhi, S. Suri, E. Welzl, Catching elephants with mice: sparse sampling for monitoring sensor networks, in: SenSys, 2007.
[39] R. Gemulla, Sampling algorithms for evolving datasets, PhD dissertation.
[40] R. Gemulla, W. Lehner, Sampling time-based sliding windows in bounded space, in: Proceedings of the 2008 ACM SIGMOD Intl. Conf. on Management of Data, pp. 379–392.
[41] P.B. Gibbons, Y. Matias, New sampling-based summary statistics for improving approximate query answers, in: Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data, 1998, pp. 331–342.
[42] P.B. Gibbons, S. Tirthapura, Distributed streams algorithms for sliding windows, in: Proceedings of the 14th Annual ACM Symposium on Parallel Algorithms and Architectures, 2002, pp. 10–13.
[43] L. Golab, D. DeHaan, E.D. Demaine, A. Lopez-Ortiz, J.I. Munro, Identifying frequent items in sliding windows over on-line packet streams, in: Proceedings of the 3rd ACM SIGCOMM Conference on Internet Measurement, 2003.
[44] L. Golab, M.T. Özsu, Processing sliding window multi-joins in continuous queries over data streams, in: Proceedings of the 29th International Conference on Very Large Data Bases, 2003, pp. 500–511.
[45] S. Guha, A. McGregor, S. Venkatasubramanian, Streaming and sublinear approximation of entropy and information distances, in: Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithm, 2006, pp. 733–742.
[46] S. Guha, A. Meyerson, N. Mishra, R. Motwani, L. O'Callaghan, Clustering data streams: Theory and practice, IEEE Trans. Knowl. Data Eng. 15 (2003).
[47] P.J. Haas, Data stream sampling: Basic techniques and results, in: M. Garofalakis, J. Gehrke, R. Rastogi (Eds.), Data Stream Management: Processing High Speed Data Streams, Springer, 2011.
[48] N. Harvey, J. Nelson, K. Onak, Sketching and streaming entropy via approximation theory, in: The 49th Annual Symposium on Foundations of Computer Science (FOCS 2008).
[49] P. Indyk, D. Woodruff, Optimal approximations of the frequency moments of data streams, in: Proceedings of the 37th Annual ACM Symposium on Theory of Computing, 2005, pp. 202–208.
[50] M. Kolonko, D. Wäsch, Sequential reservoir sampling with a nonuniform distribution, ACM Trans. Math. Software 32 (2) (2006) 257–273.
[51] H. Jowhari, M. Ghodsi, New streaming algorithms for counting triangles in graphs, in: Proceedings of the 11th COCOON, 2005, pp. 710–716.
[52] L.K. Lee, H.F. Ting, Maintaining significant stream statistics over sliding windows, in: Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithm, 2006, pp. 724–732.
[53] K. Li, Reservoir-sampling algorithms of time complexity $O(n(1+log(N/n)))$, ACM Trans. Math. Software 20 (4) (December 1994) 481–493.
[54] J. Li, D. Maier, K. Tufte, V. Papadimos, P.A. Tucker, No pane, no gain: efficient evaluation of sliding-window aggregates over data streams, in: ACM SIGMOD Record, vol. 34, no. 1, 2005.
[55] S. Muthukrishnan, Data streams: Algorithms and applications, Found. Trends Theor. Comput. Sci. 1 (2) (2005).
[56] C.R. Palmer, C. Faloutsos, Density biased sampling: an improved method for data mining and clustering, in: Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, 2000, pp. 82–92.
[57] C. Procopiuc, O. Procopiuc, Density estimation for spatial data streams, in: Proceedings of the 9th International Symposium on Spatial and Temporal Databases, 2005, pp. 109–126.
[58] M. Szegedy, The DLT priority sampling is essentially optimal, in: Proceedings of the 38th Annual ACM Symposium on Theory of Computing, 2006, pp. 150–158.
[59] N. Tatbul, S. Zdonik, Window-aware load shedding for aggregation queries over data streams, in: Proceedings of the 32nd International Conference on Very Large Data Bases, 2006.
[60] J.S. Vitter, Random sampling with a reservoir, ACM Trans. Math. Software 11 (1) (1985) 37–57.
[61] L. Zhang, Z. Li, M. Yu, Y. Wang, Y. Jiang, Random sampling algorithms for sliding windows over data streams, in: Proceedings of the 11th Joint International Computer Conference, 2005, pp. 572–575.
[62] H. Zhao, A. Lall, M. Ogihara, O. Spatscheck, J. Wang, J. Xu, A data streaming algorithm for estimating entropies of od flows, in: Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement, 2007.