SIAM J. COMPUT.
Vol. 39, No. 6, pp. 2113–2131

EFFECTIVE COMPUTATIONS ON SLIDING WINDOWS*

VLADIMIR BRAVERMAN† AND RAFAIL OSTROVSKY‡

Abstract. In the streaming model, elements arrive sequentially and can be observed only once. Maintaining statistics and aggregates is an important and nontrivial task in this model. These tasks become even more challenging in the sliding windows model, where statistics must be maintained only over the most recent n elements. In their pioneering paper, Datar et al. [*SIAM J. Comput.*, 31 (2002), pp. 1794–1813] presented the *exponential histogram*, an effective method for estimating statistics on sliding windows. In this paper we present a novel *smooth histogram* method that is more general and achieves stronger bounds than the exponential histogram. In particular, the smooth histogram method improves the approximation error rate obtained via exponential histograms. Furthermore, the smooth histogram method not only captures and improves multiple previous results on sliding windows but also extends the class of functions that can be approximated on sliding windows. In particular, we provide the first approximation algorithms for the following functions: L_p norms, frequency moments, the length of the increasing subsequence, and the geometric mean.

Key words. sliding windows, data streams, randomized algorithms, smooth histograms

AMS subject classifications. 68Q25, 68Q87, 68W25

DOI. 10.1137/090749281

1. Introduction.

Many recent applications deal with large data volumes for which methods that require multiple data passes may be infeasible. For these applications, the *data stream* model is often more appropriate. In this model, data arrives sequentially and can be observed only once. The number of data elements is unknown and may be unbounded. A typical goal is to continuously maintain statistics or aggregates over past data using minimal memory while keeping the desired precision of the answers. In this scenario, it may be challenging to maintain even simple statistics. Recently, numerous algorithms were developed for various problems in the data stream model. We refer readers to the books of Muthukrishnan [31] and Aggarwal [2] for detailed surveys on data steaming models and algorithms.

Most applications tend to discard old data and base their queries only on the recent elements. Thus, the sliding window model, in which only the last n elements are taken into consideration, is important in data stream processing. In this model we separate past elements into two groups. Recent elements represent a window of *active* or *nonexpired* elements, and the rest are *expired*. An active element may eventually become expired, but expired elements stay in this status forever. Only active elements are relevant for statistics or queries. The window can be *sequence-based*, where every insertion corresponds to a deletion of the oldest element. In *timestamp-based* windows, there is no restriction on the number of insertions and deletions. (Typically, each element is associated with a timestamp, and the window contains all elements with

*Received by the editors February 11, 2009; accepted for publication (in revised form) September 9, 2009; published electronically March 12, 2010. A preliminary version of this paper appeared in FOCS 2007, pp. 283–293.

http://www.siam.org/journals/sicomp/39-6/74928.html

†Department of Computer Science, UCLA, Los Angeles, CA 90095 (vova@cs.ucla.edu). This author's work was supported in part by NSF grants 0430254 and 0830803.

‡Department of Computer Science and Department of Mathematics, UCLA, Los Angeles, CA 90095 (rafail@cs.ucla.edu). This author's work was supported in part by an IBM Faculty Award, a Xerox Innovation Group Award, NSF grants 430254, 0716835, 0716389, 0830803, and 0916574, and a U.C. MICRO grant.

active timestamps.)

1.1. Notations. We use the following notations throughout our paper. We denote the stream by D and $p_i, i \geq 0$ as its ith element. For $0 \leq x < y$ we define $[x, y] = \{i, x \leq i \leq y\}$. *Bucket* $B(x, y)$ is the set of all stream elements between p_x and p_{y-1}: $B(x, y) = \{p_i, i \in [x, y-1]\}$. For a function f that is defined on buckets, we denote $f(i, j) = f(B(i, j))$. We denote N as the size of the stream and n as the size of a window. For the L_p problem, we denote m as the size of a vector (number of dimensions) that is presented by a stream. An algorithm maintains a $(1 \pm \epsilon)$-approximation of function f on stream D if, at any moment, the algorithm outputs f' such that $(1-\epsilon)f(D) \leq f'(D) \leq (1+\epsilon)f(D)$. Similarly, an algorithm maintains a $(1 \pm \epsilon, \delta)$-approximation of function f if at any moment the algorithm outputs f' such that $(1-\epsilon)f(D) \leq f'(D) \leq (1+\epsilon)f(D)$ with probability at least $1-\delta$. We denote $\tilde{O}(f(m)) = \frac{1}{\epsilon^{O(1)}}(\log m)^{O(1)}(\log n)^{O(1)} f(m)$. We use notation $B \subseteq_r A$ to indicate that bucket B is a suffix of A; i.e., if $A = \{p_{n_1}, \ldots, p_{n_2}\}$ (for some $n_1 < n_2$), then $B = \{p_{n_3}, \ldots, p_{n_2}\}$, where $n_1 \leq n_3 \leq n_2$. We denote by $A \cup C$ the union of adjacent buckets A and C.

1.2. Problems, results, and related work. Research on the sliding window model has a long history. In their pioneering paper, Datar et al. [17] gave effective algorithms for such fundamental statistics as count and sum of positive integers, average, $L_p, p \in [1, 2]$, etc. A further improvement to count and sum was reported by Gibbons and Tirthapura [21], who provided memory- and time-optimal algorithms for these problems. Lee and Ting [28] provided an optimal solution for a relaxed version of the counting problem, where the correct answer is provided only if it is comparable with the window's size. Besides these basic statistics, numerous problems were effectively solved on sliding windows. Chi et al. [13] considered a problem of frequent itemsets. Algorithms for frequency counts and quantiles were proposed by Arasu and Manku [4] and by Lee and Ting [29]. Datar and Muthukrishnan [18] solved problems of rarity and similarity. Babcock et al. [6] provided algorithms for variance and k-medians problems. Feigenbaum, Kannan, and Zhang [19] presented an efficient solution for the diameter of a data set in multidimensional space. Later, Chan and Sadjad [11] presented optimal solutions for this and other geometric problems. Agarwal, Har-Peled, and Varadarajan [1] used coresets to maintain statistics in the streaming model. Babcock, Datar, and Motwani [5] presented algorithms for uniform random sampling from sliding windows. In several recent papers, a variation of the streaming model is described that is different from sliding windows (see e.g., Cormode, Tirthapura, and Xu [16]; Guha, Gunopulos, and Koudas [23]; Tirthapura, Xu, and Busch [34]). In this model, data arrives in the asynchronous manner; i.e., the timestamps of arriving elements are not necessarily increasing. In this paper, as in the above papers, we do not address this model.

For more details about recent results in the sliding window model, we refer readers to the survey by Datar and Motwani in [2, Chapter 8].

The use of exponential histograms as a general technique for sliding windows was proposed by Datar et al. [17]. This method is widely used and numerous algorithms are based on exponential histograms or their variations (see [2] for examples of such applications). It is applicable to a wide class of "weakly additive" functions [2] with the following properties. Following the notations from [17], we denote by A and B adjacent buckets; we denote by $\cup$ the concatenation of adjacent buckets; and we denote by $|A|$ the number of elements in A.

1. $f(A) \geq 0$.

2. $f(A) \le poly(|A|)$.
3. $f(A \cup B) \ge f(A) + f(B)$.
4. $f(A \cup B) \le C_f(f(A) + f(B))$ for some constant $C_f \ge 1$.
5. The function $f(A)$ admits a "sketch" which requires $g(|B|)$ space and is composable; i.e., the sketch for $f(A \cup B)$ can be composed efficiently from the sketches for $f(A)$ and $f(B)$.

For this class of functions, [17] presents two general results. If f can be computed precisely on D, then it is possible to maintain f on sliding windows with relative error $(\epsilon C_f^2 + C_f - 1)$, using $O(\frac{1}{\epsilon} \log n(g + \log n))$ bits and $O(1)$ amortized time per element. Moreover, if f can be approximated on D with relative error $\hat{\epsilon}$, then f can be approximated on sliding windows with relative error $(1+\hat{\epsilon})^2 \epsilon C_f^2 + C_f - 1 + \hat{\epsilon}$ using the same space and time.

1.2.1. Summary of our results. In this paper we introduce the notion of a *smooth function* and present techniques that allow us to maintain smooth functions over sliding windows.

DEFINITION 1. *Function f is* (α, β)-smooth *if it preserves the following properties:*

1. $f(A) \ge 0$.
2. $f(A) \ge f(B)$ *for* $B \subseteq_r A$.
3. $f(A) \le poly(n)$.[1]
4. *For any* $0 < \epsilon < 1$, *there exists* $\alpha = \alpha(\epsilon, f)$ *and* $\beta = \beta(\epsilon, f)$ *such that*
   - $0 < \beta \le \alpha < 1$;
   - *if* $B \subseteq_r A$ *and* $(1-\beta)f(A) \le f(B)$, *then* $(1-\alpha)f(A \cup C) \le f(B \cup C)$ *for any adjacent* C.

Informally, smooth functions are nonnegative, nondecreasing, and polynomially bounded functions with the following property. Let $B \subseteq_r A$; that is, B contains recent elements from A and does not contain the old elements. Consider the case when $f(B)$ is close to $f(A)$. If this closeness remains no matter what elements are added to both buckets and does not depend on A and B, we say that f is smooth (for a formal definition, see section 2). To measure the closeness before and after insertions, we introduce two parameters α and β that depend only on the function f and an approximation parameter ϵ. Function f is (α, β)-smooth if, once $f(B)$ is a $(1 \pm \beta)$-approximation of $f(A)$, we can guarantee that $f(B \cup C)$ is a $(1 \pm \alpha)$-approximation of $f(A \cup C)$ for any portion of new elements C. Typically we require that $\beta \le \alpha$. Assume that there exists an algorithm that computes f precisely using g space and h time per element. (We assume that the time complexity is measured in the standard RAM model.) Our main result states that it is possible to maintain a $(1 \pm \alpha)$-approximation of f over sliding windows, using $O(\frac{1}{\beta} \log n(g + \log n))$ bits and $O(\frac{1}{\beta} h \log n)$ time. Further, a $(1 \pm \hat{\epsilon})$-approximation of f on D results in a $(1 \pm (\alpha + \hat{\epsilon}))$-approximation of f over sliding windows.

It turns out that many functions are smooth. For instance, sum, count, min, and diameter are (ϵ, ϵ)-smooth, which matches previously known results [11, 17]. More interestingly, we prove that weakly additive functions, L_p norms, frequency moments, the length of the longest subsequence, and the geometric mean are smooth. We apply our method to these functions and obtain the following results:

- We improve the general results from [17] mentioned above. For weakly ad-

[1] Similar to [17], we assume that $f(A)/f(p_N) \le poly(n)$. Otherwise, exponentially decreasing sequences could require linear memory for both smooth and exponential histograms.

ditive functions that can be computed precisely on D, the relative error is improved from $\epsilon C_f^2 + C_f - 1$ to $1 - \frac{1-\epsilon}{C_f}$. For $C_f = 1$, it gives a $(1 \pm \epsilon)$-approximation similar to [17]. For larger C_f, the ratio between relative errors is approximately C_f. One example of weakly additive functions with $C_f \gg 1$ is L_p for large p. The space and time complexities remain unchanged. For weakly additive functions that can be approximated on D, the relative error is improved from $(1+\hat{\epsilon})^2\epsilon C_f^2 + C_f - 1 + \hat{\epsilon}$ to $1 - \frac{1-\epsilon}{C_f} + \hat{\epsilon}$.

- We improve the result of [17] for $L_p, p \in [1,2]$ norms and extend it to any p. For $p \in [1,2]$ we decrease the relative error from $4\epsilon(1+\hat{\epsilon})^2+1+\hat{\epsilon}$ to $\frac{1+\epsilon}{2}+\hat{\epsilon}$, preserving memory and time. We also show that adding a multiplicative factor of $\frac{1}{\epsilon^{p-1}}$ to the memory can further decrease the relative error to ϵ. For $p > 2$ we give an optimal $(1 \pm \epsilon, \delta)$-approximation algorithm that uses $\tilde{O}(m^{1-\frac{2}{p}})$ memory, where m is the size of the estimated vector. For $p < 1$ we present a $(1 \pm \epsilon, \delta)$-approximation algorithm using $O(\frac{1}{\epsilon}\log n(\frac{1}{\epsilon^2}\log M \log\frac{\log n}{\delta\epsilon} + \log n))$ bits, where M is the maximal value of an update. For $p < 1$, L_p is not a norm; however, it is still a very useful notion of distance; see, e.g., [17, 25].
- We provide the first memory-optimal algorithm up to small factors for frequency moments over sliding windows for constant $p > 2$. The algorithm maintains a $(1 \pm \epsilon, \delta)$-approximation using $\tilde{O}(m^{1-\frac{2}{p}})$ space.
- We extend the results of Sun and Woodruff [33] to sliding windows, providing a $(1 \pm \epsilon)$-approximation of the length of the longest increasing subsequence (LIS) in sliding windows. Our algorithm uses $O(\frac{1}{\epsilon}\log n(k \log\frac{L}{k} + \log n))$ bits, where k is the length of the LIS and L is the number of distinct elements in the window.
- Geometric mean is a fundamental statistic that is useful for financial analysis (see, e.g., [14]). We provide the first $(1 \pm \epsilon)$-approximation of the geometric mean on sequence-based windows using $O(\frac{1}{\epsilon}\log n(k + \log n))$ memory. Here k is the number of bits needed to store the value of the geometric mean.

Readers may find the detailed discussion of each result in the main body of our paper.

1.2.2. High-level ideas behind our approach. Let f be (α, β)-smooth for which there exists an algorithm Λ that calculates f on D using g space and h operation per element. To maintain f on sliding windows, we construct a data structure that we call a *smooth histogram*. It consists of a set of indices $x_1 < x_2 < \cdots < x_s = N$ and instances of Λ for each bucket $B(x_i, N)$. Informally, the smooth histogram ensures the following properties of the sequence. The first two elements of the sequence always "sandwich" the window, i.e., $x_1 \le N-n < x_2$. This requirement and the monotonicity of f give us useful bounds for the sliding window W: $f(x_2, N) \le f(W) \le f(x_1, N)$. Also, f should slowly but constantly decrease with i, i.e., $f(x_{i+2}, N) < (1-\beta)f(x_i, N)$. This gradual decrease, together with the fact that f is polynomially bounded, ensures that the sequence is short, i.e., $s = O(\frac{1}{\beta}\log n)$. Finally, the values of f on successive buckets were close in the past, i.e., $f(x_{i+1}, N') \ge (1-\beta)f(x_i, N')$ for some $N' \le N$. This represents our key idea and exploits the properties of smoothness. Indeed, $f(x_2, N') \ge (1-\beta)f(x_1, N')$ for some $N' \le N$; thus, by the (α, β)-smoothness of f, we have $f(x_2, N) \ge (1-\alpha)f(x_1, N) \ge (1-\alpha)f(W)$.

To maintain a smooth histogram, we ensure that an index u becomes a successor of index $v < u$ only if at some point we have $(1-\beta)f(v, N) \le f(u, N)$. Also, we maintain the invariant $(1-\beta)f(x_i, N) > f(x_{i+2}, N)$. To do so, we check for every i

whether there exists at least one $j > i+1$ such that $(1-\beta)f(x_i, N) \leq f(x_{i+2}, N)$. If this is the case, we find maximal j and delete all intermediate indices, making x_j the successor of x_i. We also reenumerate the indices accordingly; i.e., after this step x_j will be the new x_{i+1}, x_{j+1} will be the new x_{i+2}, and so on. Note that the entire procedure requires $O(\frac{1}{\beta}h \log n)$ time since we approach every element in the list a constant number of times. (The time can be reduced to an amortized $O(h)$ when Λ supports the merging of buckets.) Similarly, we solve the case when Λ approximates f on D, although the analysis becomes slightly more complicated. We maintain s instances of Λ and s timestamps and indices; thus the space complexity is $s(g + \log n) = O(\frac{1}{\beta} \log n(g + \log n))$.

We stress that our approach works both for sequence-based and timestamp-based windows. Since memory bounds of smooth histograms are similar for both cases, we present our results for a more general model, i.e., timestamp-based windows. Interestingly, the majority of our work does not require the knowledge of actual timestamps (the only step of our algorithms that does require timestamps is checking to see whether an element has expired); the order of element arrivals is sufficient. As a result, the notion of timestamp is rarely used in our algorithms; instead we use sequence numbers. We stress that this is not a limitation of our approach, but rather a simplification of the algorithms' presentation.

Our approach is similar to exponential histograms in the sense that both methods capture gradual lessening of f using a logarithmic number of Λ instances. However, there is a critical difference between these approaches that makes our results possible. Exponential histograms divide W into distinct blocks $B_1, \ldots, B_k$. This requires a strong assumption about Λ, namely, the ability to merge buckets. Further, the algorithm [17] requires $f(B_i)$ to be close to $\sum_{j>i} f(B_j)$, and that limits applicability of exponential histograms to additive functions. Smooth histograms maintain f on suffixes rather than on distinct parts of the window and require closeness between these suffixes, eliminating the above restrictions. The ability to work with suffixes is due to the smoothness of f; thus it is a critical property.

1.2.3. An example of a smooth function. Let us illustrate the main idea by the following toy example (see also Figure 1). Consider a problem of approximating the maximum over sliding windows with precision $\epsilon = 0.5$. For the simplicity of our presentation, we assume that the stream consists of positive integers. It is easy to see that max is (ϵ, ϵ)-smooth for any ϵ. Indeed, let $A = \{p_i, \ldots, p_N\}$ for some i and let B be the suffix of A; i.e., $B = \{p_j, \ldots, p_N\}$ for some $j > i$. Let C represent a continuation of D; i.e., $C = \{p_{N+1}, \ldots, p_{N'}\}$ for some $N' > N$. It is straightforward that

$$\frac{\max(B)}{\max(A)} \leq \frac{\max(B \cup C)}{\max(A \cup C)}.$$

Indeed, if $\max(A \cup C) \neq \max(A)$, then

$$\frac{\max(B \cup C)}{\max(A \cup C)} = 1 \geq \frac{\max(B)}{\max(A)}.$$

Otherwise,

$$\frac{\max(B \cup C)}{\max(A \cup C)} = \frac{\max(B \cup C)}{\max(A)} \geq \frac{\max(B)}{\max(A)}.$$

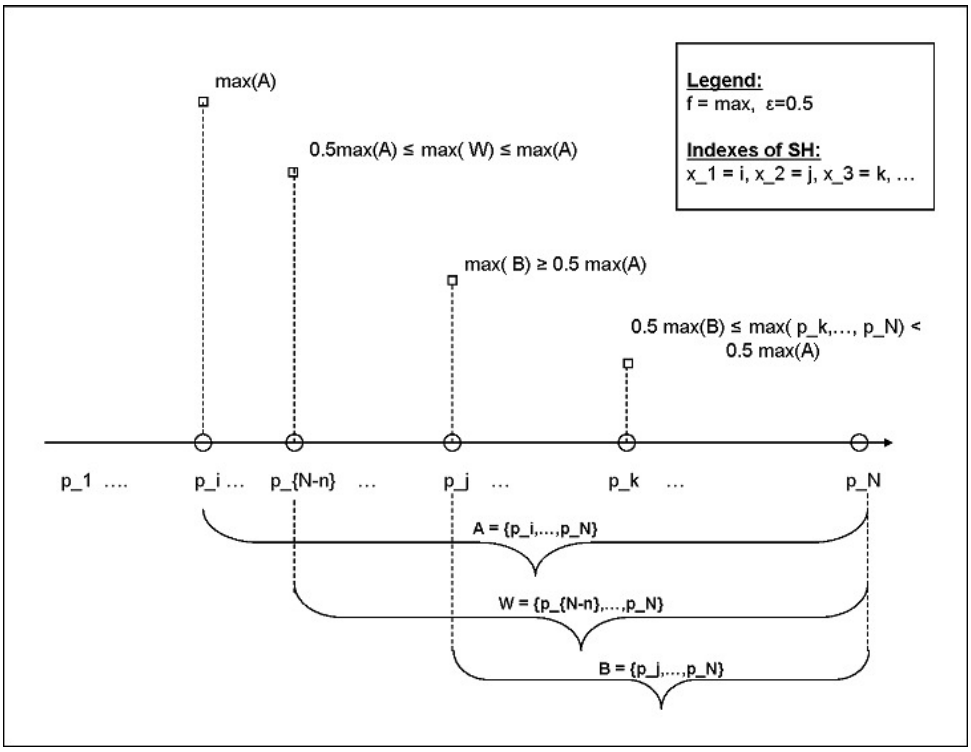

FIG. 1. *Max function.*

Thus for any ϵ, we have that if $(1-\epsilon)\max(A) \leq \max(B)$, then $(1-\epsilon)\max(A \cup C) \leq \max(B \cup C)$, i.e., max is a smooth function.

Figure 1 represents the case where the window is "sandwiched" between A and B; i.e., $i \leq N-n \leq j$. In this case, maintaining $\max(A)$ and $\max(B)$ is sufficient to obtain a 0.5-approximation; this is exactly what the smooth histogram does. To overcome the problem of expiration, the smooth histogram maintains, in addition, a sequence of indices for which max is gradually decreasing. In Figure 1 we show the first three indices of such a sequence. In this case $x_1 = i, x_2 = j$, and $x_3 = k$. Note the important condition $0.5\max(B) \leq \max\{p_k, \ldots, p_N\} < 0.5\max(A)$. That is, $\max\{p_k, \ldots, p_N\}$ cannot be used for the current approximation; however, it may be useful in the future. Indeed, eventually p_j will expire and the window will be "sandwiched" between buckets B and $\{p_k, \ldots, p_N\}$. In this case, smoothness and the fact that $0.5\max(B) \leq \max\{p_k, \ldots, p_N\}$ will ensure a proper approximation for any $N' > N$. Also, the smooth histogram ensures a gradual decrease in the value of the function for the stored indices. In Figure 1 this is reflected by the condition $\max\{p_k, \ldots, p_N\} < 0.5\max(A)$. As a result, the number of elements in this sequence is bounded by a log of the maximum of stream elements. Finally, as new elements arrive, the maximum, and thus the stored information, may be subject to change; this is achieved through our update procedure.

1.3. Roadmap. Section 2 briefly describes smooth histograms and states our main results. In section 3 we apply the smooth histogram method to approximate weakly additive functions and the L_p norms, frequency moments, the length of the

LIS, and the geometric mean. Finally, in section 4 we discuss our future work on frequency moments.

2. Smooth histograms. In this section we assume that f is (α, β)-smooth (see Definition 1) and there exists an algorithm Λ that calculates f (precisely or approximately) on the whole stream D. We construct $(1 \pm \alpha)$-approximation algorithms for such f on sliding windows. Typically, $\alpha = \epsilon$, so we obtain a $(1 \pm \epsilon)$-approximation; however, for weakly additive functions, we put $\alpha = 1 - \frac{1-\epsilon}{C_f}$. First, we assume that Λ calculates f precisely using g space and h operations per element. Λ applied on bucket $B(i, j)$ is denoted by $\Lambda(i, j)$.

DEFINITION 2. *A* smooth histogram *is a structure that consists of an increasing set of indices $X_N = \{x_1, \ldots, x_s = N\}$ and s instances of algorithm Λ, namely, $\Lambda_1, \ldots, \Lambda_s$ with the following properties:*

1. *p_{x_1} is expired, p_{x_2} is active, or $x_1 = 0$ and p_0 is active.*
2. *For all $i < s$ at least one of the following holds:*
   (a) *$x_{i+1} = x_i + 1$ and $f(x_{i+1}, N) < (1 - \beta) f(x_i, N)$.*
   (b) *$(1 - \alpha) f(x_i, N) \le f(x_{i+1}, N)$ and if $i + 2 \le s$, then $f(x_{i+2}, N) < (1 - \beta) f(x_i, N)$.*
3. *$\Lambda_i = \Lambda(x_i, N)$ maintains $f(x_i, N)$.*

LEMMA 1. *It is possible to maintain a smooth histogram using $O(\frac{1}{\beta}(g + \log n) \log n)$ bits and $O(\frac{1}{\beta} h \log n)$ operations per element. For any i, $f(x_i, N)$ can be retrieved in $O(1)$ time.*

Proof. Note that the properties 2(a) and 2(b) may overlap, so it is possible that both conditions are true for some x_i. For $N = 1$, we put $x_1 = 1, s = 1$, and we initiate Λ with p_{x_1}. Given a smooth histogram at step N and the new element p_{N+1}, we execute the following update procedure.

I. For all i, calculate $f(x_i, N + 1)$ using $\Lambda_i = \Lambda(x_i, N)$ and p_{N+1}.
II. Put $s = s + 1$ and $x_s = N + 1$ and initiate a new algorithm instance $\Lambda(N + 1, N + 1)$.
III. For $i = 1, \ldots, s - 2$ do
   (a) find the largest $j > i$ such that $f(x_j, N + 1) \ge (1 - \beta) f(x_i, N + 1)$;
   (b) delete all $x_t, i < t < j$ and all instances $\Lambda(x_t, N)$;
   (c) shift the indices accordingly:
      (i) for $j' = j, \ldots, s$ put $x_{i+j'-j+1} = x_{j'}$;
      (ii) put $s = s + i - j + 1$.
IV. Find the smallest i such that p_{x_i} is expired and $p_{x_{i+1}}$ is active. Delete all $x_j, j < i$ and Λ_j structures and change the enumeration accordingly. (It is worth mentioning that for sequence-based windows there is at most one such i; i.e., at most one element will be deleted.)

Below we prove that the update procedure maintains a smooth histogram. It follows from the last operation that property 1 is preserved. Property 3 follows from the first two steps. To prove property 2, let $v < N + 1$ be a fixed index from X_N that was not deleted during the update procedure. Let v' be the successor of v in the sequence X_N at step N; i.e., for some i we had $x_i = v, x_{i+1} = v'$.

If $v' \notin X_{N+1}$, let u and w be two successors of v in X_{N+1}. By the update procedure, it must be the case that $f(u, N+1) \ge (1-\beta) f(v, N+1) \ge (1-\alpha) f(v, N+1)$ and $f(w, N + 1) < (1 - \beta) f(v, N + 1)$. Thus, property 2(b) is correct for v.

If $v' \in X_{N+1}$ and $v' > v+1$, let $N' \le N$ be the step when v' became the successor of v. The update procedure implies that $f(v', N') \ge (1 - \beta) f(v, N')$, and since f is (α, β)-smooth we have $f(v', N+1) \ge (1-\alpha) f(v, N+1)$. Let u be the successor (if one

exists) of v' in X_{N+1}. Since v' was not deleted, we have $f(u, N+1) < (1-\beta)f(v, N+1)$. Thus, property 2(b) is correct for v.

Finally, if $v' \in X_{N+1}$ and $v' = v+1$, we have two cases. If $f(v', N+1) < (1-\beta)f(v, N+1)$, then property 2(a) is true. Otherwise, we have $f(v', N+1) \geq (1-\beta)f(v, N+1) \geq (1-\alpha)f(v, N+1)$. Let u be the successor (if one exists) of v' in X_{N+1}. Similarly, $f(u, N+1) < (1-\beta)f(v, N+1)$. Thus, property 2(b) is correct for v, and property 2 is preserved for any $v < N+1$.

Let us bound the size s of the sequence X_N. By the properties above, we have that for any i either $f(x_{i+2}, N)$ or $f(x_{i+1}, N)$ is less then $(1-\beta)f(x_i, N)$. This and the fact that f is polynomially bounded imply $s = O(\frac{1}{\beta}\log n)$. Since we maintain exactly s instances of algorithm Λ and timestamps, the space complexity is $O(s(g + \log n)) = O(\frac{1}{\beta}(g + \log n)\log n)$, and the time complexity per element is $O(sh) = O(\frac{1}{\beta}h \log n)$. ∎

THEOREM 1. *Let f be an (α, β)-smooth function. If there exists an algorithm Λ that precisely calculates f on streams, uses space g, and performs h operations per element, then there exists an algorithm Λ' that calculates a $(1 \pm \alpha)$-approximation of f on sliding windows and uses $O(\frac{1}{\beta}(g + \log n)\log n)$ bits and $O(\frac{1}{\beta}h \log n)$ operations per element.*

Proof. The algorithm maintains a smooth histogram and outputs $f(x_2, N)$ as an approximation of f on the window. To prove that this is a $(1 \pm \alpha)$-approximation, let j be the index of the last active element, so the precise value is $f(j, N)$. If property 2(a) is correct for x_1, then, by property 1, $j = x_2$ and the answer is precise. Otherwise property 2(b) is correct for x_1, and we have, since f is monotonic, $f(j, N) \geq f(x_2, N) \geq (1-\alpha)f(x_1, N) \geq (1-\alpha)f(j, N)$. ∎

In many cases it is impossible to calculate f precisely. Below we show how to adapt approximation algorithms to sliding windows. We assume that Λ maintains a $(1 \pm \hat{\epsilon})$-approximation of f on D, $\hat{\epsilon} \leq \frac{\beta}{4}$, and uses $g(\hat{\epsilon})$ space and $h(\hat{\epsilon})$ operations per element. We call such an approximation f'.

DEFINITION 3. *The approximate smooth histogram is a structure that consists of an increasing set of indices $X_N = \{x_1, \ldots, x_s = N\}$ and s instances of algorithm Λ, namely, $\Lambda_1, \ldots, \Lambda_s$ with the following properties:*

1. *p_{x_1} is expired, p_{x_2} is active, or $x_1 = 0$.*
2. *For all $i < s$, one of the following holds:*
   (a) *$x_{i+1} = x_i + 1$ and $f'(x_{i+1}, N) < (1 - \frac{\beta}{2})f'(x_i, N)$.*
   (b) *$(1-\alpha)f(x_i, N) \leq f(x_{i+1}, N)$ and if $i+2 \leq s$, then $f'(x_{i+2}, N) < (1 - \frac{\beta}{2})f'(x_i, N)$.*
3. *$\Lambda_i = \Lambda(x_i, N)$ maintains $f'(x_i, N)$.*

LEMMA 2. *It is possible to maintain an approximate smooth histogram using $O(\frac{1}{\beta}(g(\hat{\epsilon}) + \log n)\log n)$ bits and $O(\frac{1}{\beta}h(\hat{\epsilon})\log n)$ operations per element.*

Proof. The update procedure repeats Lemma 1. For $N = 1$, we put $x_1 = 1, s = 1$, and we initiate Λ with p_{x_1}. Given an approximate smooth histogram at step N and the new element p_{N+1}, we execute the following update procedure.

I. For all i, calculate $f'(x_i, N+1)$ using $\Lambda_i = \Lambda(x_i, N)$ and p_{N+1}.
II. Put $s = s+1$ and $x_s = N+1$, and initiate a new algorithm instance $\Lambda(N+1, N+1)$.
III. For $i = 1, \ldots, s-2$ do
   (a) find the largest $j > i$ such that $f'(x_j, N+1) \geq (1 - \frac{\beta}{2})f'(x_i, N+1)$;
   (b) delete all $x_t, i < t < j$ and all instances $\Lambda(x_t, N)$;
   (c) shift the indices accordingly:

(i) for $j' = j, \dots, s$ put $x_{i+j'-j+1} = x_{j'}$;
(ii) put $s = s + i - j + 1$.

IV. Find the smallest i such that p_{x_i} is expired and $p_{x_{i+1}}$ is active. Delete all $x_j, j < i$ and Λ_j structures, and change the enumeration accordingly.

It follows from the last operation that property 1 is preserved. Property 3 follows from the first two steps. To prove property 2, let $v < N+1$ be a fixed index from X_N that was not deleted during the update procedure. Let v' be the successor of v in the sequence X_N at step N; i.e., for some i we had $x_i = v, x_{i+1} = v'$.

If $v' \notin X_{N+1}$, let u and w be two successors of v in X_{N+1}. By the update procedure, it must be the case that $f'(u, N+1) \geq (1 - \frac{\beta}{2}) f'(v, N+1)$, thus

$$(1-\alpha) f(v, N+1) \leq (1-\beta) f(v, N+1) \leq \left(1 - \frac{\beta}{2}\right) \frac{1 - \frac{\beta}{4}}{1 + \frac{\beta}{4}} f(v, N+1)$$

$$\leq \frac{1 - \frac{\beta}{2}}{1 + \frac{\beta}{4}} f'(v, N+1) \leq \frac{f'(u, N+1)}{1 + \frac{\beta}{4}} \leq f(u, N+1).$$

Also, it must be the case that $f'(w, N+1) < (1 - \frac{\beta}{2}) f'(v, N+1)$; thus, property 2(b) is correct for v.

If $v' \in X_{N+1}$ and $v' > v+1$, let $N' \leq N$ be the step when v' became the successor of v. The update procedure implies that $f'(v', N') \geq (1 - \frac{\beta}{2}) f'(v, N')$ and thus

$$(1-\beta) f(v, N') \leq \frac{(1-\beta)}{(1 - \frac{\beta}{4})} f'(v, N') \leq \frac{(1-\beta)}{(1 - \frac{\beta}{4})(1 - \frac{\beta}{2})} f'(v', N')$$
$$\leq \frac{(1-\beta)(1 + \frac{\beta}{4})}{(1 - \frac{\beta}{4})(1 - \frac{\beta}{2})} f(v', N') \leq f(v', N').$$

Since f is (α, β)-smooth we have $f(v', N+1) \geq (1-\alpha) f(v, N+1)$. Let u be the successor (if one exists) of v' in X_{N+1}. Since v' was not deleted, we have $f'(u, N+1) < (1 - \frac{\beta}{2}) f'(v, N+1)$. Thus, property 2(b) is correct for v.

Finally, if $v' \in X_{N+1}$ and $v' = v+1$, we have two cases. If $f'(v', N+1) < (1 - \frac{\beta}{2}) f'(v, N+1)$, then property 2(a) is true. Otherwise, we have $f(v', N+1) \geq (1-\alpha) f(v, N+1)$, repeating the calculations above. Let u be the successor of v' in X_{N+1}. Similarly, $f'(u, N+1) < (1 - \frac{\beta}{2}) f'(v, N+1)$. Thus, property 2(b) is correct for v. Therefore property 2 is preserved for any $v < N+1$.

Properties of a histogram imply that for any i, either $f'(x_{i+2}, N)$ or $f'(x_{i+1}, N)$ is less than $(1 - \frac{\beta}{2}) f'(x_i, N)$. This property and the facts that f' is at least a $(1 \pm \frac{\beta}{4})$-approximation of f and f is polynomially bounded, imply $s = O(\frac{1}{\beta} \log n)$. Since we maintain exactly s instances of algorithm Λ, the space complexity is thus $O(sg) = O(\frac{1}{\beta}(g(\hat{\epsilon}) + \log n) \log n)$, and the time complexity per element is $O(sh) = O(\frac{1}{\beta} h \log n)$. □

THEOREM 2. *Let f be an (α, β)-smooth function (see Definition 1). If there exists an algorithm Λ that maintains a $(1 \pm \hat{\epsilon})$-approximation of f on D, using space $g(\hat{\epsilon})$ and performing $h(\hat{\epsilon})$ operations per stream element, then there exists an algorithm Λ' that maintains a $(1 \pm (\alpha + \hat{\epsilon}))$-approximation of f on sliding windows and uses $O(\frac{1}{\beta}(g(\hat{\epsilon}) + \log n) \log n)$ bits and $O(\frac{1}{\beta} h(\hat{\epsilon}) \log n)$ operations per element.*

Proof. The algorithm maintains an approximate smooth histogram and outputs $f'(x_2, N)$ as an approximation of f on the window. Let j be the index of the last

active element, so the precise value is $f(j,N)$. If property 2(a) is correct for x_1, then by property 1, $j = x_2$ and the answer is a $(1\pm\hat{\epsilon})$-approximation of $f(j,N)$. Otherwise property 2(b) is correct for x_1, and we have

$$(1+\alpha+\hat{\epsilon})f(j,N) \geq \frac{1+\alpha+\hat{\epsilon}}{1+\frac{\beta}{4}} f'(x_2,N) \geq f'(x_2,N).$$

Also,

$$(1-\alpha-\hat{\epsilon})f(j,N) \leq (1-\alpha-\hat{\epsilon})f(x_1,N) \leq (1-\hat{\epsilon})f(x_2,N) \leq f'(x_2,N). \qquad \square$$

Similarly, we can approximate functions for which there exists an algorithm Λ that maintains a $(1\pm\hat{\epsilon},\delta)$-approximation on D. The proof remains the same; we need only ensure that the probability of failure is at most δ. Recall that the smooth histogram uses $O(\frac{1}{\beta}\log n)$ instances of Λ. Thus, if for each instance we limit the probability of failure by $\frac{\delta\beta}{\log n}$, then by the union bound the total probability of failure will be at most δ. We obtain the following theorem.

THEOREM 3. *Let f be an (α,β)-smooth function. If there exists an algorithm Λ that maintains a $(1\pm\hat{\epsilon},\delta)$-approximation of f on D using space $g(\hat{\epsilon},\delta)$ and performing $h(\hat{\epsilon},\delta)$ operations per stream element, then there exists an algorithm Λ' that maintains a $(1\pm(\alpha+\hat{\epsilon}))$-approximation of f on sliding windows and uses $O(\frac{1}{\beta}(g(\hat{\epsilon},\frac{\delta\beta}{\log n})+\log n)\log n)$ bits and $O(\frac{1}{\beta}h(\hat{\epsilon},\frac{\delta\beta}{\log n})\log n)$ operations per element.*

Note that the proofs above are correct for sequence-based and timestamp-based windows.

3. Applications.

3.1. Weakly additive functions. Let f be any weakly additive function that can be precisely computed on D using space g and time h. The authors in [17] proved that f can be approximated on sliding windows with relative error $\epsilon C_f^2 + C_f - 1$, space $O(\frac{1}{\epsilon}(g+\log n)\log n)$, and amortized time $O(h)$. Smooth histograms improve the relative error, preserving space and time complexities.

LEMMA 3. *Weakly additive function f with parameter C_f is $(1-\frac{1-\epsilon}{C_f},\epsilon)$-smooth.*

Proof. Let A be a bucket and let B be its suffix such that $(1-\epsilon)f(A) \leq f(B)$. For any adjacent bucket C we have

$$\left(1-\left(1-\frac{1-\epsilon}{C_f}\right)\right)f(A\cup C) \leq (1-\epsilon)(f(A)+f(C)) \leq f(B)+f(C) \leq f(B\cup C). \qquad \square$$

COROLLARY 1. *A weakly additive function f can be approximated on sliding windows with relative error $(1-\frac{1-\epsilon}{C_f})$ using $O(\frac{1}{\epsilon}(g+\log n)\log n)$ memory bits and $O(h)$ amortized time per element.*

Proof. By applying Theorem 1, we almost obtain the result. The only problem is the logarithmic number of operations per element. This can be reduced using the fact that sketches are composable. Instead of recalculating f on buckets for each new element, we do it for every $(\frac{1}{\epsilon}\log n)$th element, collecting all new elements in an auxiliary buffer. Let v be the index of the first collected point. Since sketches are composable, we can compute $f(u,N)$ using the sketch for $f(u+1,N)$ and p_u for any $v \leq u < N$. Using the sketches for all of the $f(u,N)$, we compute $f(x_i,N)$ for all i. The rest of the algorithm remains unchanged. $\square$

Also, [17] showed that given an algorithm that maintains a $(1 \pm \hat{\epsilon})$-approximation of f on D, f can be approximated on sliding windows with relative error $(1+\hat{\epsilon})^2\epsilon C_f^2 + C_f - 1 + \hat{\epsilon}$, using the same space and time. Using Theorem 2 and repeating the arguments from Corollary 1, we obtain the following corollary.

COROLLARY 2. *There exists an algorithm that maintains an approximation of f on sliding windows with relative error $(1 - \frac{1-\epsilon}{C_f} + \hat{\epsilon})$ using $O(\frac{1}{\epsilon}(g + \log n)\log n)$ space and $O(h)$ amortized time per element.*

Note that $1 - \frac{1-\epsilon}{C_f} \leq \epsilon C_f^2 + C_f - 1$ and $1 - \frac{1-\epsilon}{C_f} + \hat{\epsilon} \leq (1+\hat{\epsilon})^2\epsilon C_f^2 + C_f - 1 + \hat{\epsilon}$. The relative improvement is comparable with C_f; thus the improvement becomes significant for large C_f.

3.2. L_p norms. For $p \geq 0$ and a vector $V = \langle v_1, \ldots, v_m \rangle$, define $L_p(V) = (\sum_{i=1}^m |v_i|^p)^{\frac{1}{p}}$. In the streaming model V is represented as follows. Each element of a stream D is a pair (i, a), where $i \in [m]$ and $a \in [-M, M]$ for some positive M. The value of the ith coordinate is given by the summation $v_i = \sum_{(i,a)\in D} a$. If m is small, it is easy to calculate the L_p norm simply by maintaining the value of each coordinate. However, the usual assumption is that m is large and $\Omega(m)$ space is not allowed.

Computing the frequency moments is a fundamental problem that is directly related to L_p norms. In this paper we use the definition that was presented by Bhuvanagiri et al. [8]: $F_p = \sum_{i=1}^m |v_i|^p = L_p^p$. In many papers the simpler model is considered, where $a = 1$ for all pairs (i, a) and a's are omitted.

The first algorithms for frequency moments were proposed in the seminal paper of Alon, Matias, and Szegedy [3]. For $p = 0, 2$ they provided $(1 \pm \epsilon, \delta)$-approximation algorithms that use only polylogarithmic memory. For $p > 2$ they presented an algorithm that uses $O(m^{1-\frac{1}{p}})$ memory and showed a lower bound of $\Omega(m^{1-\frac{5}{p}})$. For $p > 2$, numerous improvements to lower and upper bounds were reported, including the work of Bar-Yossef et al. [7], Chakrabarti, Khot, and Sun [10], Coppersmith and Kumar [15], and Ganguly [20]. Finally, Indyk and Woodruff [26] and later Bhuvanagiri et al. [8] presented algorithms that use $\tilde{O}(m^{1-\frac{2}{p}})$ memory and are optimal. The last two algorithms can be used as well for approximation of $L_p, p > 2$ norms. For $p \in [0, 2]$ Indyk [25] and Kane, Nelson, and Woodruff [27] presented algorithms that maintain L_p norms using polylogarithmic space.

The extension of these problems to sliding windows is straightforward. At any moment $L_p(W) = (\sum_{i=1}^m v_i^p)^{\frac{1}{p}}$, where $v_i = \sum_{(i,a)\in W} a$ and W is the current window. Similar to [17], we restrict a to be positive. For negative a, [17] showed that even for $p = 1$ and $m = 1$, the lower bound on the memory is $\Omega(n)$. The only known result for L_p norms was presented in [17] for $p \in [1, 2]$. The algorithm maintains L_p with high probability and relative error $4\epsilon(1+\epsilon)^2 + 1 + \epsilon$ using $O(\frac{1}{\epsilon}\log N(\log N + \log M \log(1/\delta)/\epsilon^2))$ bits.

We extend this result to any p and provide a better approximation ratio for $p \in [1, 2]$. We prove that L_p is $(\epsilon, \frac{\epsilon^p}{p})$-smooth for $p \geq 1$ and (ϵ, ϵ)-smooth for $p < 1$, so our method can be applied. For $p > 2$, we apply the algorithm from [8] to show an optimal $(1 \pm \epsilon, \delta)$-approximation algorithm using $\tilde{O}(m^{1-\frac{2}{p}})$ bits. For $L_p, p < 1$, we use the algorithm from [25] to construct a $(1 \pm \epsilon, \delta)$-approximation algorithm using $O(\frac{1}{\epsilon}\log n(\frac{1}{\epsilon^2}\log M \log\frac{\log n}{\delta\epsilon} + \log n))$ bits. Finally, for $L_p, 1 \leq p \leq 2$, we improve [17] by decreasing the relative error from $4(1+\hat{\epsilon})^2\epsilon + 1 + \hat{\epsilon} = 1 + \epsilon'$ to $\frac{1+\epsilon}{2} + \hat{\epsilon} = \frac{1}{2} + \epsilon''$ for some ϵ', ϵ''. Our method uses $O(\frac{1}{\epsilon}\log n(\frac{1}{\epsilon^2}\log M \log\frac{\log n}{\delta\epsilon} + \log n))$ memory bits. We exploit the fact that L_p^p is weakly additive with $C_f \leq 2$ and thus is $(\frac{1+\epsilon}{2}, \epsilon)$-smooth. However,

it is also $(\epsilon, \frac{\epsilon^p}{p})$-smooth and, alternatively, we can achieve a $(1 \pm \epsilon)$-approximation using $O(\frac{1}{\epsilon^p} \log n(\frac{1}{\epsilon^2} \log M \log \frac{\log n}{\delta\epsilon} + \log n))$ bits, i.e., by increasing memory usage by the factor $\frac{1}{\epsilon^{p-1}}$.

One may argue that for $p > 2$, L_p^p is also a weakly additive function, so we can apply the exponential histograms method. While this is true, note that the relative error that can be achieved using exponential histograms is $\epsilon C_f^2 + C_f - 1$ (see [17]). For large p the relative error becomes significantly larger then 1.

Our results for the L_p norm can be directly applied to frequency moments. Thus, for constant $p > 2$ we obtain optimal results for frequency moments.

In the remainder of this section we show how smooth histograms can be used to approximate L_p norms on sliding windows. Recall that for this problem D represents a vector $V = \langle v_1, \ldots, v_m \rangle$. Each element is a pair (i, a), where $i \in [m]$ and $a \in [M]$ for some positive $M \leq n^{O(1)}$. The value of the ith coordinate is given by $v_i = \sum_{(i,a) \in W} a$, and $L_p(W)$ is defined as $(\sum_{i=1}^m v_i^p)^{\frac{1}{p}}$. For this model Datar et al. [17] showed a $(1 \pm (4\epsilon(1+\epsilon)^2 + 1 + \epsilon), \delta)$-approximation for $p \in [1, 2]$. We extend this result to any p and improve the approximation parameter for $p \in [1, 2]$. Below we prove that L_p is a smooth function.

LEMMA 4. *For $p \geq 1$, L_p is an $(\epsilon, \frac{\epsilon^p}{p})$-smooth function. For $p < 1$, L_p is an (ϵ, ϵ)-smooth function.*

Proof. It is easy to see that in this model L_p satisfies the properties of function f, i.e., it is monotonic on buckets, polynomially bounded, and positive. Let $p > 1$ and let V, Y be vectors that are represented by buckets A, B ($B \subseteq_r A$) such that $(1 - \frac{\epsilon^p}{p}) L_p(A) \leq L_p(B)$. Recall that in our model, $v_i \geq y_i$ for all $i \in [m]$. We have

$$(1 - \epsilon^p)\|V\|_p^p \leq \left(1 - \frac{\epsilon^p}{p}\right)^p \|V\|_p^p \leq \|Y\|_p^p.$$

By the triangle inequality, $\|V\|_p^p \geq \|Y\|_p^p + \|V - Y\|_p^p$ and thus $\|V - Y\|_p \leq \epsilon\|V\|_p \leq \epsilon\|V + Z\|_p$ for any Z that is represented by adjacent bucket C (recall that $z_i \geq 0$). By the triangle inequality,

$$\|V + Z\|_p \leq \|Y + Z\|_p + \|V - Y\|_p \leq \|Y + Z\|_p + \epsilon\|V + Z\|_p.$$

That concludes the lemma for $p > 1$. For $p < 1$, let V, Y be vectors such that $(1 - \epsilon)\|V\| \leq \|Y\|$. We have the following for any $Z = \langle z_1, \ldots, z_m \rangle$:

$$\|Y + Z\|_p^p = \|Y\|_p^p + \sum_{i=1}^m ((y_i + z_i)^p - y_i^p) \geq (1 - \epsilon)^p \|V\|_p^p$$

$$+ (1 - \epsilon)^p \sum_{i=1}^m ((v_i + z_i)^p - v_i^p) = (1 - \epsilon)^p \|V + Z\|_p^p.$$

The inequality follows from the assumption that $\|Y\|_p \geq (1 - \epsilon)\|V\|_p$ and the fact that function $f(x) = (x + a)^p - x^p$ is decreasing for $x \geq 0$ and $p < 1, a > 0$. ∎

For $p > 2$ we apply the algorithm of Bhuvanagiri et al. [8] for frequency moments. In our model, frequency moments are simply $L_p(x) = F_p(x)^{\frac{1}{p}}$. Thus, for $p > 1$, the $(1 \pm \epsilon)$-approximation for F_p is also the $(1 \pm \epsilon)$-approximation for L_p. Recall their result.

THEOREM 4 (see [8]). *There exists an algorithm that computes a $(1 \pm \epsilon, \frac{3}{4})$-approximation of $L_p, p > 2$ using $O(\frac{p^2}{\epsilon^{2+\frac{4}{p}}} m^{1-\frac{2}{p}} \log^2 N(\log m + \log N))$ bits.*

COROLLARY 3. *There exists an algorithm that uses* $\tilde{O}(m^{1-\frac{2}{p}})$ *memory and calculates a* $(1 \pm \epsilon, \delta)$*-approximation of the* $L_p, p > 2$ *norm over sliding windows.*

Proof. We apply Theorem 3 and the algorithm from [8] (after amplification). The resulting space complexity is $O(\frac{p^3}{\epsilon^{2p+\frac{4}{p}}} m^{1-\frac{2}{p}} \log^3 n (\log m + \log n) \log \frac{\log n}{\delta\epsilon})$. ∎

For $p < 1$, similar to [17], we can apply the algorithm of Indyk [25]. Recall that this algorithm provides a $(1 \pm \epsilon, \delta)$-approximation of $L_p(D), p \in (0, 2]$ and uses $O(\frac{1}{\epsilon^2} \log M \log \frac{1}{\delta})$ bits for each sketch. Additionally, $O(\frac{1}{\epsilon^2} \log M \log \frac{m}{\delta} \log \frac{1}{\delta})$ bits are required and are common for all sketches. Since $L_p(W)$ is an (ϵ, ϵ)-smooth function, we can apply Theorem 3 using the algorithm above.

COROLLARY 4. *There exists an algorithm that calculates a* $(1 \pm \epsilon, \delta)$*-approximation of the* $L_p, p < 1$ *norm on sliding windows and uses space* $O(\frac{1}{\epsilon} \log n (\frac{1}{\epsilon^2} \log M \log \frac{\log n}{\delta\epsilon} + \log n))$.

For $p \in [1, 2]$ we present two solutions. We can repeat the arguments above or we can apply Corollary 1 noting that $L_p^p, p \in [1, 2]$ is weakly additive with $C_f \leq 2$ by [17].

COROLLARY 5. *It is possible to maintain an approximation of* $L_p, p \in [1, 2]$ *over sliding windows with relative error* $\epsilon + \frac{\epsilon^p}{p}$ *and probability at least* $1 - \delta$ *using* $O(\frac{1}{\epsilon^p} \log n (\frac{1}{\epsilon^2} \log M \log \frac{\log n}{\delta\epsilon} + \log n))$ *bits. Also, it is possible to obtain a relative error of* $\frac{1+}{2} + \epsilon$ *using* $O(\frac{1}{\epsilon} \log n (\frac{1}{\epsilon^2} \log M \log \frac{\log n}{\delta\epsilon} + \log n))$ *bits.*

The second result strictly improves [17], while the first one significantly improves the relative error and increases memory requirements by a factor of $\frac{1}{\epsilon^{p-1}}$.

3.3. Frequency moments. Approximation for frequency moments can be derived immediately from section 3.2. Thus, this section has rather illustrative purposes; we do show, however, that frequency moments are smooth.

LEMMA 5. *For* $p \geq 1$, F_p *is an* $(\epsilon, \frac{\epsilon^p}{p^p})$*-smooth function. For* $p < 1$, F_p *is an* (ϵ, ϵ)*-smooth function.*

Proof. Let $p \geq 1$ and let X, Y be two vectors such that

$$\left(1 - \frac{\epsilon^p}{p^p}\right) \sum_{i=1}^{m} x_i^p \leq \sum_{i=1}^{m} y_i^p.$$

We have

$$\sum_{i=1}^{m} (x_i - y_i)^p \leq \frac{\epsilon^p}{p^p} \sum_{i=1}^{m} x_i^p.$$

Thus for any vector Z that is added to X and Y, we have

$$\left(\sum_{i=1}^{m} (x_i - y_i)^p\right)^{\frac{1}{p}} \leq \frac{\epsilon}{p} \left(\sum_{i=1}^{m} (x_i + z_i)^p\right)^{\frac{1}{p}},$$

and by the triangle inequality,

$$\left(\sum_{i=1}^{m} (y_i + z_i)^p\right)^{\frac{1}{p}} \geq \left(\sum_{i=1}^{m} (x_i + z_i)^p\right)^{\frac{1}{p}}$$
$$- \left(\sum_{i=1}^{m} (x_i - y_i)^p\right)^{\frac{1}{p}} \geq \left(1 - \frac{\epsilon}{p}\right) \left(\sum_{i=1}^{m} (x_i + z_i)^p\right)^{\frac{1}{p}}.$$

Thus,

$$\sum_{i=1}^{m}(y_i+z_i)^p \geq \left(1-\frac{\epsilon}{p}\right)^p \sum_{i=1}^{m}(x_i+z_i)^p \geq (1-\epsilon)\sum_{i=1}^{m}(x_i+z_i)^p.$$

If $p<1$ we have the following, for X,Y,Z as above, and assuming $(1-\epsilon)F_p(X) \leq F_p(Y)$:

$$\sum_{i=1}^{m}(y_i+z_i)^p = \sum_{i=1}^{m}y_i^p + \sum_{i=1}^{m}((y_i+z_i)^p - y_i^p) \geq (1-\epsilon)\sum_{i=1}^{m}x_i^p$$
$$+(1-\epsilon)\sum_{i=1}^{m}((x_i+z_i)^p - x_i^p) = (1-\epsilon)\sum_{i=1}^{m}(x_i+z_i)^p.$$

The inequality follows from the assumption above and from the fact that function $(x+a)^p - x^p$ is decreasing for $p<1, a>0$. ∎

Now, we can apply the algorithm of Bhuvanagiri et al. [8].

COROLLARY 6. *For constant $p>2$, there exists an algorithm that uses $\tilde{O}(m^{1-\frac{2}{p}})$ memory and calculates a $(1\pm\epsilon,\delta)$-approximation of the pth frequency moment over sliding windows.*

Proof. The proof is identical to that of Corollary 3. ∎

3.4. Length of the longest increasing subsequence. Let D be a stream where p_i is an integer, $p_i \in [L]$ for some L. An increasing subsequence is defined as $p_{x_1},\ldots,p_{x_k}$ such that $x_i < x_{i+1}$ and $p_{x_i} \leq p_{x_{i+1}}$ for $i<k$. (In fact, the sequence is nondecreasing, but we follow the notations of the previous works.) The longest increasing subsequence $LIS(D)$ is an increasing subsequence with maximal size k. Correspondingly, $LIS(W)$ on window W is defined as a LIS on the set of last n elements. This is a well-studied statistic that is used in bioinformatics and other fields. (See the works of Gusfield [24] and Pevzner [32].) Recent results in the streaming model include the papers by Liben-Nowell, Vee, and Zhu [30], Gopalan et al. [22], and Sun and Woodruff [33]. The last paper presents a memory-optimal algorithm that uses $\Theta(k\log\frac{L}{k})$ memory. For sliding windows Chen, Yang, and Yuan [12] present an algorithm that uses $\Omega(n)$ memory.

We extend the result of Sun and Woodruff [33] to sliding windows. Our algorithm uses $O(\frac{1}{\epsilon}\log n(k\log\frac{L}{k}+\log n))$ bits and provides a $(1\pm\epsilon)$-approximation of the length of the LIS. First, we show that our technique is applicable.

LEMMA 6. *The LIS length is an (ϵ,ϵ)-smooth function.*

Proof. Let $Y \subseteq_r X$ be two buckets such that

$$(1-\epsilon)|LIS(X)| \leq |LIS(Y)|. \tag{1}$$

It is enough to show that for any new element q we guarantee that

$$(1-\epsilon)|LIS(X\cup\{q\})| \leq |LIS(Y\cup\{q\})|. \tag{2}$$

To prove (2) we establish the following useful facts. First, we show that if adding q increases $LIS(Y)$, then (2) holds. Assume that this is the case, i.e.,

$$|LIS(Y\cup\{q\})| > |LIS(Y)|. \tag{3}$$

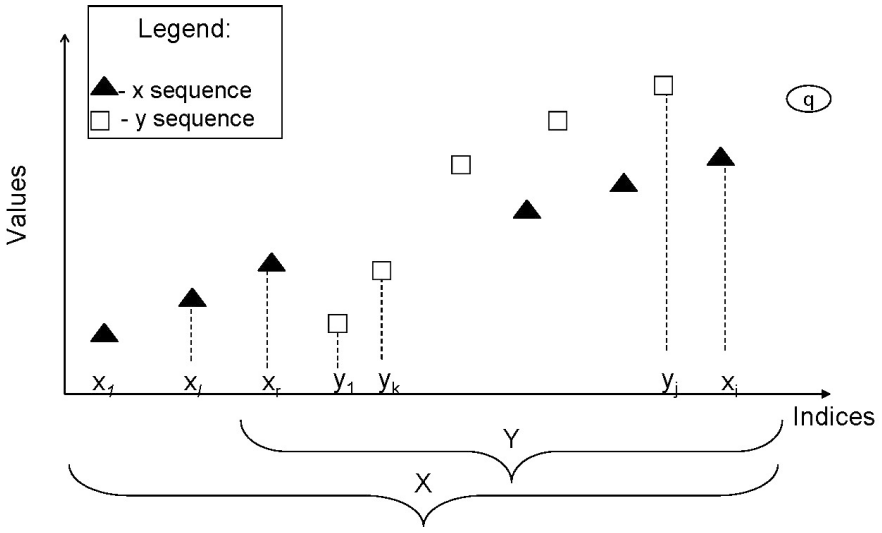

FIG. 2. *Proof of Lemma* 6.

By adding a single element, a LIS can increase by at most one. In particular, (3) implies that $|LIS(Y \cup \{q\})| = |LIS(Y)| + 1$. Thus,

$$(1-\epsilon)|LIS(X \cup \{q\})| \leq (1-\epsilon)(|LIS(X)|+1) \leq (1-\epsilon)|LIS(X)|+1$$

$$\leq |LIS(Y)|+1 = |LIS(Y \cup \{q\})|.$$

Here the last inequality follows from (1). Also, if $|LIS(X \cup \{q\})| = |LIS(X)|$, then (2) obviously holds:

$$(1-\epsilon)|LIS(X \cup \{q\})| = (1-\epsilon)|LIS(X)| \leq |LIS(Y)| \leq |LIS(Y \cup \{q\})|.$$

In the remainder of the proof, we show that no other cases are possible. Assume, toward a contradiction, that

$$(|LIS(Y \cup \{q\})| = |LIS(Y)|) \wedge (|LIS(X \cup \{q\})| > |LIS(X)|). \tag{4}$$

The assumption (4) implies several facts about the LIS's on X and Y. In particular, there exists a LIS on X that can be extended by adding q. Let $x_1, \ldots, x_i$ be indices of one of such (extendable) LIS's on X, i.e.,

$$p_{x_i} \leq q. \tag{5}$$

Let $y_1, \ldots, y_j$ be indices of a LIS on Y. Intuitively, we show that these two sequences must "intersect" (see Figure 2 for the illustration). Thus, we can construct a new LIS on Y by concatenating the y-sequence before the intersection with the x-sequence after the intersection. By (5), this new LIS can be extended by adding q. This contradicts assumption (4).

To formally prove the intuition, we establish some useful facts about the sequences $x_1, \ldots, x_i$ and $y_1, \ldots, y_j$. Figure 2 illustrates these facts and the relation between these two sequences. Assumption (4) implies that any longest extendable increasing

sequence on X must end before any longest increasing sequence on Y and have a larger last value; i.e.,

$$(x_i > y_j) \wedge (p_{x_i} < p_{y_j}). \tag{6}$$

Indeed, assume that $p_{x_i} \geq p_{y_j}$. Then, and by (5), we have $q \geq p_{x_i} \geq p_{y_j}$. Thus $p_{y_1}, \ldots, p_{y_j}, q$ is an increasing subsequence; i.e., $|LIS(Y \cup \{q\})| > |LIS(Y)|$, which contradicts assumption (4). Thus, it must be the case that $p_{x_i} < p_{y_j}$, in particular $x_i \neq y_j$. Finally, if $x_i < y_j$, then $p_{x_1}, \ldots, p_{x_i}, p_{y_j}$ is an increasing subsequence on X that contradicts the maximality of $p_{x_1}, \ldots, p_{x_i}$. Thus, under assumption (4), claim (6) is correct.

Let x_l be the largest index such that $p_{x_l} \notin Y$. There exists such an index since otherwise the entire subsequence $p_{x_1}, \ldots, p_{x_i}$ belongs to Y; i.e., $|LIS(Y)| = |LIS(X)|$. But then $p_{x_1}, \ldots, p_{x_i}, q$ is an increasing subsequence on $Y \cup \{q\}$; i.e., $|LIS(Y \cup \{q\})| \geq i + 1 > |LIS(Y)|$, which contradicts (4). Next, we show that

$$p_{x_l} > p_{y_1}. \tag{7}$$

Indeed, $|LIS(Y)| = j \geq i - l$ since $p_{x_{l+1}}, \ldots, p_{x_i}$ is an increasing subsequence on Y. If $j = i - l$, then $|LIS(Y \cup \{q\})| > |LIS(Y)|$ since $p_{x_{l+1}}, \ldots, p_{x_i}, q$ is the increasing subsequence on $LIS(Y \cup \{q\})$; thus $j > i - l$. Also, if $p_{x_l} \leq p_{y_1}$, then $p_{x_1}, \ldots, p_{x_l}, p_{y_1}, \ldots, p_{y_j}$ is an increasing subsequence on X with size larger than $i = |LIS(X)|$. Thus (7) is correct.

For $s \in [j]$ define $next(s) = \min_{x_t \in [i]}\{x_t \geq y_s\}$. Note that this quantity is well defined since $next(s) \leq x_i$ by (6). We say that y_s is *covered* if $p_{next(s)} \geq p_{y_s}$. We know that y_1 is covered because $next(1) \geq y_1 > x_l$ and thus by (7) $p_{next(1)} \geq p_{x_l} > p_{y_1}$. Let y_k be the maximal covered index and denote $r = next(k) - 1$. Note that $r \geq l \geq 1$ and, by (6), $k < j$. We claim that

$$p_{x_r} \leq p_{y_{k+1}}. \tag{8}$$

Indeed, $next(k+1) > y_k > x_r$ and, by the maximality of k, we have $p_{y_{k+1}} > p_{next(k+1)} \geq p_{x_r}$. Now consider two sequences $p_{x_1}, \ldots, p_{x_r}, p_{y_{k+1}}, \ldots, p_{y_j}$ and $p_{y_1}, \ldots, p_{y_k}, p_{r+1}, \ldots, p_{x_i}$. By the definition of *next* and by (8), both of these sequences are increasing. Thus, it must be the case that $p_{y_{k+1}}, \ldots, p_{y_j}$ and $p_{r+1}, \ldots, p_{x_i}$ have the same length (otherwise we could increase one of the LIS's). Thus, $p_{y_1}, \ldots, p_{y_k}, p_{r+1}, \ldots, p_{x_i}$ is the longest increased subsequence. But then we can extend it by adding q. This contradicts (4) and concludes our proof. ∎

Therefore, we can apply Theorem 1 using the algorithm of Sun and Woodruff [33].

COROLLARY 7. *Let D be a steam of integers such that $p_i \in [L]$. There exists an algorithm that computes a $(1 \pm \epsilon)$-approximation of $|LIS(W)|$, where W is the current window. The algorithm uses space*

$$O\left(\frac{1}{\epsilon} \log n \left(|LIS(W)| \log \frac{L}{|LIS(W)|} + \log n\right)\right).$$

3.5. Geometric mean. The smooth histogram method may be extended to some nonsmooth functions. To illustrate this point, we discuss the problem of the geometric mean. Let D be a stream of positive real polynomially bounded numbers. The geometric mean is defined as $GM(D) = (\prod_{i=1}^{N} p_i)^{\frac{1}{N}}$. For sliding windows we

define the geometric mean as $GM(W) = (\prod_{i=N-n}^{N} p_i)^{\frac{1}{n}}$. To the best of our knowledge, this problem has not been considered in the sliding windows model. One possible method is to apply [17] $\log p_i$ to a stream of logarithms of the stream elements. The problem of this approach is that the resulting stream may contain both positive and negative values. Since the exponential histogram method [17] is applicable to streams with nonnegative values, this method cannot be directly applied to estimate the geometric mean. In fact, smooth histograms cannot be applied directly as well, since $GM(W)$ is not necessarily a nondecreasing function.

We thus employ a novel idea of "splitting" the sliding window into two subsets, $W_{\geq 1}$ and $W_{<1}$. The subset $W_{\geq 1}$ represents all active elements $p_i \geq 1$, and $W_{<1}$ represents all active elements $p_i < 1$. It is easy to see that both $GM(W_{\geq 1})$ and $\frac{1}{GM(W_{<1})}$ are smooth functions, and thus we can apply the smooth histogram method. It is worth mentioning that our idea of splitting is also applicable to exponential histograms. As a result, we present the first approximation algorithm for the geometric mean on sequence-based sliding windows that uses $O\left(\frac{1}{\epsilon}\log n(k+\log n)\right)$ space, where k is the number of bits needed to store the answer.

COROLLARY 8. *There exists an algorithm that computes a $(1\pm\epsilon)$-approximation of the geometric mean over sequence-based sliding windows using $O\left(\frac{1}{\epsilon}(k+\log n)\log n\right)$ space.*

Proof. We divide D into two substreams, $D_{<1} = \{p_i | p_i \leq 1\}$ and $D_{>1} = \{p_i | p_i \geq 1\}$. The active window W is correspondingly separated into $W_{<1}$ and $W_{>1}$. We define two auxiliary functions on buckets $h(B) = (\prod_{i\in B} p_i)^{\frac{1}{n}}$ and $g(B) = (\prod_{i\in B} \frac{1}{p_i})^{\frac{1}{n}}$. We apply our method on these two substreams and then combine the results to obtain the approximation. Note that n is fixed, but $W_{>1}, W_{<1}$ are timestamp-based windows.

It is easy to see that h is (ϵ,ϵ)-smooth on $D_{>1}$. Indeed, it is monotonic (for fixed n), polynomially bounded, and new elements do not change the ratio $\frac{h(A)}{h(B)}$ for any buckets A, B. Thus, we can maintain a $(1\pm\epsilon)$-approximation of $h(W) = GM(W_{>1})$ using $O\left(\frac{1}{\epsilon}(k+\log n)\log n\right)$ memory. Similarly, g is an (ϵ,ϵ)-smooth function on $D_{<1}$, and thus we obtain a $(1\pm\epsilon)$-approximation of $g(W) = \frac{1}{GM(W_{<1})}$. Dividing h by g, we obtain a $(1\pm 2\epsilon)$-approximation of the geometrical mean. ∎

4. Future work. In this paper we prove that F_p is $(\epsilon, \frac{\epsilon^p}{p^p})$-smooth, and thus smooth histograms can be used to approximate frequency moments with $\tilde{O}(p^p m^{1-\frac{2}{p}})$ bits. Thus, for constant $p > 2$ we obtain optimal results. However, the additional factor of p^p makes the smooth histogram approach infeasible for large p, which the current paper does not handle. In a recent work [9], we showed how to handle frequency moments for arbitrary p. In particular, we showed how to compute F_p using $\tilde{O}(m^{1-\frac{1}{p}})$ bits for any $p > 2$.

Also, we point out the following property of the smooth histogram (this property is, in fact, shared by the previous work, e.g., [17]). Let f be a function that can be approximated on sliding windows using the smooth histogram method. Then the smooth histogram for sliding window W provides "complimentary" approximation of f for any "subwindow" $W' \subseteq_r W$. Indeed, for any W' there exists i such that $p_{x_i} \notin W'$ and $p_{x_{i+1}} \in W'$. By the properties of the smooth histogram, this implies that $f(\{p_{x_i}, \ldots, p_N\})$ is an approximation of $f(W')$. This property may be useful for applications since it allows changing a window's size in the online manner and still preserving statistics without requiring recomputation. Moreover, a smooth histogram in fact stores approximations for all suffixes of W, and thus it allows us to track changes in f as the window shortens. Further elaboration of this property may be an

interesting research direction.

Acknowledgment. We thank the anonymous referees for their helpful suggestions.

REFERENCES

[1] P. K. Agarwal, S. Har-Peled, and K. R. Varadarajan, *Geometric Approximation via Coresets*, in Combinatorial and Computational Geometry, Math. Sci. Res. Inst. Publ. 52, Cambridge University Press, Cambridge, UK, 2005, pp. 1–30.

[2] C. C. Aggarwal, *Data Streams: Models and Algorithms (Advances in Database Systems)*, Springer-Verlag, New York, 2006.

[3] N. Alon, Y. Matias, and M. Szegedy, *The space complexity of approximating the frequency moments*, J. Comput. System Sci., 58 (1999), pp. 137–147.

[4] A. Arasu and G. S. Manku, *Approximate counts and quantiles over sliding windows*, in Proceedings of the 23rd ACM SIGMOD-SIGART Symposium on Principles of Database Systems, Paris, 2004, pp. 286–296.

[5] B. Babcock, M. Datar, and R. Motwani, *Sampling from a moving window over streaming data*, in Proceedings of the 13th ACM-SIAM Symposium on Discrete Algorithms, San Francisco, 2002, pp. 633–634.

[6] B. Babcock, M. Datar, R. Motwani, and 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, San Diego, 2003, pp. 234–243.

[7] Z. Bar-Yossef, T. S. Jayram, R. Kumar, and D. Sivakumar, *An information statistics approach to data stream and communication complexity*, J. Comput. System Sci., 68 (2004), pp. 702–732.

[8] L. Bhuvanagiri, S. Ganguly, D. Kesh, and C. Saha, *Simpler algorithm for estimating frequency moments of data streams*, in Proceedings of the 17th ACM-SIAM Symposium on Discrete Algorithms, Miami, 2006, pp. 708–713.

[9] V. Braverman, R. Ostrovsky, and C. Zaniolo, *Optimal sampling from sliding windows*, in Proceedings of the 28th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, Providence, 2009, pp. 147–156.

[10] A. Chakrabarti, S. Khot, and X. Sun, *Near-optimal lower bounds on the multi-party communication complexity of set disjointness*, in Proceedings of the 18th IEEE Conference on Computational Complexity, Aarhus, Denmark, 2003, pp. 107–117.

[11] T. M. Chan and B. S. Sadjad, *Geometric optimization problems over sliding windows*, Internat. J. Comput. Geom. Appl., 16 (2006), pp. 145–158.

[12] E. Chen, L. Yang, and H. Yuan, *Longest increasing subsequences in windows based on canonical antichain partition*, Theoret. Comput. Sci., 378 (2007), pp. 223–236.

[13] Y. Chi, H. Wang, P. S. Yu, and R. R. Muntz, *Moment: Maintaining closed frequent itemsets over a stream sliding window*, in Proceedings of the Fourth IEEE International Conference on Data Mining, Brighton, UK, 2004, pp. 59–66.

[14] I. Cooper, *Arithmetic versus geometric mean estimators: Setting discount rates for capital budgeting*, European Financial Management, 2 (1996), pp. 157–167.

[15] D. Coppersmith and R. Kumar, *An improved data stream algorithm for frequency moments*, in Proceedings of the 15th ACM-SIAM Symposium on Discrete Algorithms, New Orleans, 2004, pp. 151–156.

[16] G. Cormode, S. Tirthapura, and B. Xu, *Time-decaying sketches for sensor data aggregation*, in Proceedings of the 26th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Beijing, 2007, pp. 215–224.

[17] M. Datar, A. Gionis, P. Indyk, and R. Motwani, *Maintaining stream statistics over sliding windows*, SIAM J. Comput., 31 (2002), pp. 1794–1813.

[18] M. Datar and S. Muthukrishnan, *Estimating rarity and similarity over data stream windows*, in Proceedings of the 10th European Symposium on Algorithms, Rome, 2002, pp. 323–334.

[19] J. Feigenbaum, S. Kannan, and J. Zhang, *Computing diameter in the streaming and sliding-window models*, Algorithmica, 41 (2004), pp. 25–41.

[20] S. Ganguly, *Estimating frequency moments of data streams using random linear combinations*, in Proceedings of the APPROX-RANDOM 2004, Cambridge, MA, 2004, pp. 369–380.

[21] P. B. Gibbons and S. Tirthapura, *Distributed streams algorithms for sliding windows*, in Proceedings of the 14th ACM Symposium on Parallel Algorithms and Architectures, Winnipeg, Manitoba, Canada, 2002, pp. 63–72.

[22] P. Gopalan, T. S. Jayram, R. Krauthgamer, and R. Kumar, *Estimating the sortedness of a data stream*, in Proceedings of the 18th ACM-SIAM Symposium on Discrete Algorithms 2007, New Orleans, pp. 318–327.

[23] S. Guha, D. Gunopulos, and N. Koudas, *Correlating synchronous and asynchronous data streams*, in Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, 2003, pp. 529–534.

[24] D. Gusfield, *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*, Cambridge University Press, Cambridge, UK, 1997.

[25] P. Indyk, *Stable distributions, pseudorandom generators, embeddings, and data stream computation*, J. ACM, 53 (2006), pp. 307–323.

[26] P. Indyk and D. Woodruff, *Optimal approximations of the frequency moments of data streams*, in Proceedings of the 37th ACM Symposium on Theory of Computing, Baltimore, 2005, pp. 202–208.

[27] D. M. Kane, J. Nelson, and D. P. Woodruff, *Revisiting Norm Estimation in Data Streams*, http://www.mit.edu/∼minilek/papers/norms.pdf (2008).

[28] L. K. Lee and H. F. Ting, *Maintaining significant stream statistics over sliding windows*, in Proceedings of the 17th ACM-SIAM Symposium on Discrete Algorithms, Miami, 2006, pp. 724–732.

[29] L. K. Lee and H. F. Ting, *A simpler and more efficient deterministic scheme for finding frequent items over sliding windows*, in Proceedings of the 25th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, Chicago, 2006, pp. 290–297.

[30] D. Liben-Nowell, E. Vee, and A. Zhu, *Finding longest increasing and common subsequences in streaming data*, J. Comb. Optim., 11 (2003), p. 2006.

[31] S. Muthukrishnan, *Data streams: Algorithms and applications*, Found. Trends Theor. Comput. Sci., 1 (2005), pp. 117–236.

[32] P. A. Pevzner, *Computational Molecular Biology*, MIT Press, Cambridge, MA, 2000.

[33] X. Sun and D. P. Woodruff, *The communication and streaming complexity of computing the longest common and increasing subsequences*, in Proceedings of the 18th ACM-SIAM Symposium on Discrete Algorithms, New Orleans, 2007, pp. 336–345.

[34] S. Tirthapura, B. Xu, and C. Busch, *Sketching asynchronous streams over a sliding window*, in Proceedings of the 25th ACM Symposium on Principles of Distributed Computing, Denver, 2006, pp. 82–91.