Efficient Summing over Sliding Windows

Ran Ben Basat[1], Gil Einziger[2], Roy Friedman[3], and Yaron Kassner[4]

1 **Department of Computer Science, Technion, Haifa, Israel**
sran@cs.technion.ac.il
2 **Department of Computer Science, Technion, Haifa, Israel**
gilga@cs.technion.ac.il
3 **Department of Computer Science, Technion, Haifa, Israel**
roy@cs.technion.ac.il
4 **Department of Computer Science, Technion, Haifa, Israel**
kassnery@cs.technion.ac.il

Abstract

This paper considers the problem of maintaining statistic aggregates over the last W elements of a data stream. First, the problem of counting the number of 1's in the last W bits of a binary stream is considered. A lower bound of $\Omega(\frac{1}{\epsilon}+\log W)$ memory bits for $W\epsilon$-*additive approximations* is derived. This is followed by an algorithm whose memory consumption is $O(\frac{1}{\epsilon}+\log W)$ bits, indicating that the algorithm is optimal and that the bound is tight. Next, the more general problem of maintaining a *sum* of the last W integers, each in the range of $\{0,1,\ldots,R\}$, is addressed. The paper shows that approximating the sum within an *additive error* of $RW\epsilon$ can also be done using $\Theta(\frac{1}{\epsilon}+\log W)$ bits for $\epsilon=\Omega\left(\frac{1}{W}\right)$. For $\epsilon=o\left(\frac{1}{W}\right)$, we present a *succinct* algorithm which uses $\mathcal{B}\cdot(1+o(1))$ bits, where $\mathcal{B}=\Theta\left(W\log\left(\frac{1}{W\epsilon}\right)\right)$ is the derived lower bound. We show that all lower bounds generalize to randomized algorithms as well. All algorithms process new elements and answer queries in $O(1)$ worst-case time.

1998 ACM Subject Classification E.1 [Data Structures] Lists, stacks, and queues

Keywords and phrases Streaming, Statistics, Lower Bounds

Digital Object Identifier 10.4230/LIPIcs.SWAT.2016.11

1 Introduction

Background

The ability to process and maintain statistics about large streams of data is useful in many domains, such as security, networking, sensor networks, economics, business intelligence, etc. Since the data may change considerably over time, there is often a need to keep the statistics only with respect to some window of the last W elements at any given point. A naive solution to this problem is to keep the W most recent elements, add an element to the statistic when it arrives, and subtract it when it leaves the window. Yet, when the window of interest is large, which is often the case when data arrive at high rate, the required memory overhead may become a performance bottleneck.

Though it may be tempting to think that RAM memory is cheap, a closer look indicates that there are still performance benefits in maintaining small data structures. For example, hardware devices such as network switches prefer to store important data in the faster and scarcely available SRAM than in DRAM. This is in order to keep up with the ever increasing line-speed of modern networks. Similarly, on a CPU, caches provide much faster performance

than DRAM memory. Thus, small data structures that fit inside a single cache line and can possibly be pinned there are likely to result in much faster performance than a solution that spans multiple lines that are less likely to be constantly maintained in the cache.

A well known method to conserve space is to approximate the statistics. Basic-Counting is one of the most basic textbook examples of such approximated stream processing problems [12]. In this problem, one is required to keep track of the number of 1's in a stream of binary bits. A $(1+\epsilon)$-*multiplicative approximation* algorithm for this problem using $O\left(\frac{1}{\epsilon}\log^2 W\epsilon\right)$ bits was shown in [12]. This solution works with amortized $O(1)$ time, but its worst case time complexity is $O(\log W)$.

A more practical related problem is Basic-Summing, in which the goal is to maintain the sum of the last W elements. When all elements are non-negative integers in the range $[R+1]=\{0,1,\ldots,R\}$, the work in [12] naturally extends to provide a $(1+\epsilon)$-multiplicative approximation of this problem using $O\left(\frac{1}{\epsilon}\cdot\left(\log^2 W+\log R\cdot(\log W+\log\log R)\right)\right)$ bits. The amortized time complexity becomes $O(\frac{\log R}{\log W})$ and the worst case is $O(\log W+\log R)$.

Our Contributions

In this paper, we explore the benefits of changing the approximation guarantee from *multiplicative* to *additive*. With a multiplicative approximation, the result returned can be different from the correct one by at most a multiplicative factor, e.g., 5%. On the other hand, in an additive approximation, the absolute error is bounded, e.g., a deviation of up to ± 5. When the expected number of ones in a stream is small, multiplicative approximation is more appealing, since its absolute error is small. However, in this case, an accurate (sparse) representation can be even more space efficient than the multiplicative approximation. On the other hand, when many ones are expected, additive approximation gives similar outcomes to multiplicative approximation. Furthermore, the potential space saving becomes significant in this case, motivating our exploration.

Our initial contribution is a formally proved memory lower bound of $\Omega(\frac{1}{\epsilon}+\log W)$ for $W\epsilon$-additive approximations for the Basic-Counting problem.

Our second contribution is a space optimal algorithm providing a $W\epsilon$-*additive approximation* for the Basic-Counting problem. It consumes $O(\frac{1}{\epsilon}+\log W)$ memory bits with a worst case time complexity of $O(1)$, matching the lower bound.

Next, we explore the more general Basic-Summing problem. Here, the results are split based on the value of ϵ. Specifically, our third contribution is an (asymptotically) space optimal algorithm providing an $RW\epsilon$-additive approximation for the Basic-Summing problem when $\epsilon^{-1}\leq 2W\left(1-\frac{1}{\log W}\right)$.[1] It uses $O(\frac{1}{\epsilon}+\log W)$ memory bits and has $O(1)$ worst case time complexity. For other values of ϵ, we show a lower bound of $\Omega(W\log\left(\frac{1}{W\epsilon}\right))$ and a corresponding algorithm requiring $O\left(W\log\left(\frac{1}{2W\epsilon}+1\right)\right)$ memory bits with $O(1)$ worst case time complexity. Furthermore, we show that this algorithm is succinct for $\epsilon=o(W^{-1})$, i.e. its space requirement is only (1+o(1)) times the lower bound.

To get a feel for the applicability of these results, consider for example an algorithmic trader that makes transactions based on a moving average of the gold price. He samples the spot price once every millisecond, and wishes to approximate the average price for the last hour, i.e., $W=3.6\cdot 10^6$ samples. The current price is around \$1200, and with a standard deviation of \$10, he safely assumes the price is bounded by $R\triangleq 1500$. The trader is willing to withstand an error of 0.1%, which is approximately \$1.2. Our algorithm provides a

[1] In this paper, the logarithms are of base 2 and the o(1) notation is for $W\to\infty$.

$WR\epsilon_A$ (the 'A' stands for *Additive*) additive-approximation using $\left(\frac{1}{2\epsilon_A} + 2\log W\right)(1 + o(1))$ memory bits, while the algorithm by Datar et al. [12] computes a multiplicative $(1 + \epsilon_M)$ (the 'M' stands for *Multiplicative*) approximation using $\left\lceil \frac{1}{2\epsilon_M} + 1 \right\rceil \lceil \log(2WR\epsilon_M + 1) + 1 \rceil$ buckets of size $\lceil \log W + \log(\log W + \log R) \rceil$ bits each. Using our algorithm, the trader sets $\epsilon_A = R^{-1} = \frac{1}{1500}$, which guarantees that as long as the price of gold stays above \$1000, the error remains lower than required. The multiplicative approximation algorithm requires setting $\epsilon_M = 0.1\%$, and uses $501 \cdot \lceil \log(1080001) + 1 \rceil = 12525$ buckets of size 27 bits each and about $41KB$ overall. In comparison, our algorithm with the parameters above requires only about 100 bytes.

Another useful application for our algorithm is counting within a *fixed* additive error. The straight-forward algorithm for solving Basic-Counting uses a W-bits array which stores the entire window, replacing the oldest recorded bit with a new one whenever such arrives. Assume a ± 5 error is allowed. Using the multiplicative-approximation algorithms, one has to set $\epsilon_M = \frac{5}{W}$, which requires more than W bits, worse than exact counting. In contrast, setting $\epsilon_A = \frac{5}{W}$ for our algorithm reduces the memory consumption of the exact solution by nearly 90%.

In summary, we show that additive approximations offer significant space reduction opportunities. They can be obtained with a constant worst case time complexity, which is important in real-time and time sensitive applications.

2 Related Work

In [12], Datar et al. first presented the problem of counting the number of 1's in a sliding window of size W over a binary stream, and its generalization to summing a window over a stream of integers in the range $\{0, 1, \ldots, R\}$. They have introduced a data structure called *exponential histogram* (*EH*). *EH* is a time-stamp based structure that partitions the stream into *buckets*, saving the time elapsed since the last 1 in the bucket was seen. Using *EH*, they have derived a space-optimal algorithm for approximating Basic-Summing within a multiplicative-factor of $(1+\epsilon)$, which uses $O\left(\frac{1}{\epsilon}\log^2 W + \log R \cdot (\log W + \log\log R)\right)$ memory bits. The structure allows estimating a class of aggregate functions such as counting, summing and computing the ℓ_1 and ℓ_2 norms of a sliding window in a stream containing integers. The exponential histogram technique was later expanded [3] to support computation of additional functions such as k-median and variance. Gibbons and Tirthapura [13] presented a different structure called *waves*, which improved the worst-case runtime of processing a new element to a constant, keeping space requirement comparable when $R = poly(W)$. Braverman and Ostrowsky [7] defined *smooth histogram*, a generalization of the exponential histogram, which allowed estimation of a wider class of aggregate functions and improved previous results for several functions such as l_p norms and frequency moments. Lee and Ting [15] presented an improved algorithm, requiring less space if a $(1 + \epsilon)$ approximation is guaranteed only when the ones consist of a significant fraction of the window. They also presented the λ counter [16] that counts bits over a sliding window as part of a frequent items algorithm. Our design is more space efficient as it requires $O(\frac{1}{\varepsilon} + log(n))$ bits instead of $O(\frac{1}{\varepsilon} \cdot log(n))$ bits.

In [8], Cohen and Strauss considered a generalization of the bit-counting problem on a sliding window for computing a weighted sum for some decay function, such that the more recent bits have higher weights. Cormode and Yi [9] solved bit counting in a distributed setting with optimal communication between nodes. Table 1 and Table 2 summarize previous works on the Basic-Counting and Basic-Summing problems and compare them to our own algorithms.

Basic-Counting	Approximation Guarantee	Memory Requirement	Amortized Addition Time	Worst-Case Addition Time	Maximal Additive Error
Datar et al. [12]	$(1+\epsilon)$-Multiplicative	$O\left(\frac{1}{\epsilon}\log^2 W\epsilon\right)$	$O(1)$	$O(\log W)$	$W\epsilon$
Gibbons and Tirthapura [13]	$(1+\epsilon)$-Multiplicative	$O\left(\frac{1}{\epsilon}\log^2 W\epsilon\right)$	$O(1)$	$O(1)$	$W\epsilon$
Lee and Ting [15]	$(1+\epsilon)$-Multiplicative, whenever there are at least θW 1-bits	$O\left(\frac{1}{\epsilon}\log^2\frac{1}{\theta} + \log W\theta\epsilon\right)$	$O(1)$	$O(1)$	$W\epsilon$
This Paper	$W\epsilon$-Additive	$O\left(\frac{1}{\epsilon}+\log W\epsilon\right)$	$O(1)$	$O(1)$	$W\epsilon$

Table 1 Comparison of Basic-Counting Algorithms.

Basic-Summing	Approximation Guarantee	Memory Requirement	Amortized Addition Time	Worst-Case Addition Time	Maximal Additive Error
Datar et al. [12]	$(1+\epsilon)$-Multiplicative	$O\left(\frac{1}{\epsilon}\left(\log^2 W + \log R\log W + \log R\log\log R\right)\right)$	$O\left(\frac{\log R}{\log W}\right)$	$O(\log W + \log R)$	$RW\epsilon$
Gibbons and Tirthapura [13]	$(1+\epsilon)$-Multiplicative	$O\left(\frac{1}{\epsilon}\left(\log W+\log R\right)^2\right)$	$O(1)$	$O(1)$	$RW\epsilon$
This Paper	$RW\epsilon$-Additive for $\epsilon \geq \frac{1}{2W}$	$O\left(\frac{1}{\epsilon}+\log W\right)$	$O(1)$	$O(1)$	$RW\epsilon$
	$RW\epsilon$-Additive for $\epsilon \leq \frac{1}{2W}$	$O\left(W\cdot\log\left(\frac{1}{W\epsilon}\right)\right)$			

Table 2 Comparison of Basic-Summing Algorithms.

Extensive studies were conducted on many other streaming problems over sliding windows such as Top-K [18, 20], Top-K tuples [22], Quantiles [2], heavy hitters [5, 6, 14], distinct items [24], duplicates [21], Longest Increasing Subsequences [7, 1], Bloom filters [17, 19], graph problems [10, 11] and more.

3 Basic-Counting Problem

▶ **Definition 1** (Approximation). Given a value V and a constant ϵ, we say that $\widehat{V}$ is an ϵ-*multiplicative approximation* of V if $|V-\widehat{V}| < \epsilon V$. We say that $\widehat{V}$ is an ϵ-*additive approximation* of V if $|V-\widehat{V}| < \epsilon$.

▶ **Definition 2** (Basic-Counting). Given a stream of bits and a parameter W, maintain the number of 1's in the last W bits of the stream. Denote this number by C^W.

3.1 Lower Bound

We now show lower bounds for the memory requirement for approximating Basic-Counting.

▶ **Lemma 3.** *For any ϵ and W, any deterministic algorithm that provides a $W\epsilon$-additive approximation for* Basic-Counting *requires at least* $\left\lfloor \frac{W}{\lfloor 2W\epsilon+1 \rfloor} \right\rfloor \geq \left\lfloor \frac{1}{2\epsilon+W^{-1}} \right\rfloor$ *bits.*

Proof. Denote $z \triangleq \left\lfloor \frac{W}{\lfloor 2W\epsilon+1 \rfloor} \right\rfloor$. We prove the lemma by showing 2^z arrangements that must lead to different configurations. Consider the language of all concatenations of z blocks of size $\lfloor 2W\epsilon+1 \rfloor$, such that each block consists of only ones or only zeros:

$$L_{W,\epsilon} = \{w_0 w_1 \cdots w_{z-1} \mid \forall j \in [z] : w_j = 0^{\lfloor 2W\epsilon+1 \rfloor} \vee w_j = 1^{\lfloor 2W\epsilon+1 \rfloor}\}$$

Assume, by way of contradiction, that two different words

$$s^1 = w_0^1 w_1^1 \cdots w_{z-1}^1, s^2 = w_0^2 w_1^2 \cdots w_{z-1}^2 \in L_{W,\epsilon}$$

lead the algorithm to the same configuration. Denote the index of the last block that differs between s^1 and s^2 by $t \triangleq max\{\tau \mid w_\tau^1 \neq w_\tau^2\}$. Next, consider the sequences $s^1 \cdot 0^{(t-1)\lfloor 2W\epsilon+1 \rfloor}$ and $s^2 \cdot 0^{(t-1)\lfloor 2W\epsilon+1 \rfloor}$. The algorithm must reach the same configuration after processing these sequences, even though the number of ones differs by $\lfloor 2W\epsilon+1 \rfloor > 2W\epsilon$. Therefore, the algorithm's error must be greater than $W\epsilon$ at least for one of the sequences, in contradiction to the assumption. We have shown 2^z words that lead to different configurations and therefore any deterministic algorithm that provides $\epsilon - additive$ approximation to Basic-Counting must have at least z bits of state. ◀

An immediate corollary of Lemma 3 is that any exact algorithm for Basic-Counting requires at least W bits, i.e., the naive solution is optimal. We next establish a second lower bound, which is useful for proving that our algorithm, presented below, is space optimal up to a constant factor.

▶ **Lemma 4.** *Fix some $\epsilon \leq \frac{1}{4}$. Any deterministic algorithm that provides a $W\epsilon$-additive approximation for the* Basic-Counting *problem requires at least $\lfloor \log W \rfloor$ bits.*

Proof. Assume that some algorithm A gives a $W\epsilon$-additive approximation using m memory bits. Consider A's run on the sequence $s = 0^W \cdot 1^{2^m}$. Since A is using m bits, it reaches some memory configuration c at least twice after processing the zeros in the sequence. Assume that A first reached c after seeing $0^W \cdot 1^y$ (where $y < 2^m$). This means that A must output some number $a_c \leq y + W\epsilon$ if queried. Now assume A returns to configuration c after reading z additional ones. This means A will return to c after every additional sequence of z ones. Therefore, for every integer q, after processing the sequence $0^W \cdot 1^{y+qz}$, A will reach configuration c. We can then pick a large q (such that $y + qz \geq W$), which means that the query answer for configuration c, a_c, has to be at least $W(1-\epsilon)$, as the window is now all-ones. We get $W(1-\epsilon) \leq a_c \leq y + W\epsilon$ and thus $2^m > y \geq W(1-2\epsilon)$. Putting everything together, we conclude that $m > \log(W(1-2\epsilon)) = \log W + \log(1-2\epsilon) \geq \log W - 1$, for $\epsilon \leq \frac{1}{4}$. Finally, since m is an integer, this implies $m \geq \lfloor \log W \rfloor$. ◀

▶ **Theorem 5.** *Let $\epsilon \leq \frac{1}{4}$. Any deterministic algorithm that provides a $W\epsilon$-additive approximation for the* Basic-Counting *problem requires at least* $\left\lfloor \max\left\{\log W, \frac{1}{2\epsilon+W^{-1}}\right\}\right\rfloor$ *bits.*

Proof. Immediate from lemmas 3 and 4. ◀

Finally, we extend our lower bounds to randomized algorithms.

▶ **Theorem 6.** *Let $\epsilon \leq \frac{1}{4}$. Any randomized Las Vegas algorithm that provides a $W\epsilon$-additive approximation for the* Basic-Counting *problem requires at least $\left\lfloor \max\left\{\log W, \frac{1}{2\epsilon+W^{-1}}\right\}\right\rfloor$ bits. Further, for any fixed $\delta \in (0, 1/2)$, any Monte Carlo algorithm that with probability at least $1-\delta$ approximates* Basic-Counting *within $W\epsilon$ error at any time instant, requires $\Omega(\frac{1}{\epsilon} + \log W)$ bits.*

Proof. We say that algorithm A is ϵ-correct on a input instance S if it is able to approximate the number of 1's in the last W bits, at every time instant while reading S, to within an additive error of $W\epsilon$.

We remind the reader that in our case, a Las Vegas (LV) algorithm for the Basic-Counting approximation problem is a randomized algorithm which is *always* ϵ-correct. In contrast, a Monte Carlo (MC) algorithm is a randomized procedure that is allowed to provide approximation with error larger than $W\epsilon$, with probability at most δ.

The Yao Minimax principle [23] implies that the amount of memory required for a deterministic algorithm to approximate a random input chosen according to a distribution $\mathbf{p}$ is a lower bound on the expected space consumption of a Las Vegas algorithm for the worst input. To prove a $\left\lfloor \frac{1}{2\epsilon+W^{-1}} \right\rfloor$ lower bound, we consider padding the language $L_{W,\epsilon}$ which is defined in Lemma 3. Specifically, we define $\mathbf{p}$ as the uniform distribution over all inputs in the language

$$L_{LV} = L_{W,\epsilon} \cdot \left\{0^W\right\}.$$

That is, the input consist of all bit sequences in $L_{W,\epsilon}$, followed by a sequence of W zeros. The trailing 0's are used to force the algorithm to reach distinct configurations after reading the first W input bits. As implied by the lemma, any deterministic algorithm which is always correct for a random instance requires at least $\left\lfloor \frac{1}{2\epsilon+W^{-1}} \right\rfloor$ bits, as it has to arrive to a distinct state for each input $s \in L_{W,\epsilon}$. The argument for a lower bound of $\lfloor \log W \rfloor$ bits is similar.

Next, we use the Minimax principle analogue for Monte Carlo algorithms [23], which states that for any input distribution $\mathbf{p}$ and $\delta \in [0, 1/2]$, any randomized algorithm that is always (for any input) ϵ-correct with probability at least $1-\delta$ uses in expectation at least half as much memory as the optimal deterministic algorithm that errs (i.e., is not ϵ-correct) with probability at most 2δ on a random instance drawn according to $\mathbf{p}$. Once again, we consider $\mathbf{p}$ to be the uniform distribution over

$$L_{MC} = L_{W,\epsilon} \cdot \left\{0^W\right\}.$$

Since the distribution is uniform, any deterministic algorithm, which is ϵ-correct with probability at least $1-2\delta$ on a random instance drawn according to $\mathbf{p}$, is actually ϵ-correct on $1-2\delta$ fraction of the inputs. Similar to the LV case, the argument in Lemma 3 implies that the algorithm must reach a distinct configuration after reading the first W bits of each of the $(1-2\delta)\cdot|L_{MC}|$ inputs it is ϵ-correct on. Consequently, the algorithm must use at least $\log\left((1-2\delta)\cdot|L_{MC}|\right)$ bits of memory. Applying the Minimax principle, the derived lower bound B_{MC} for any MC algorithm is:

$$B_{MC} \geq \frac{1}{2}\log\left((1-2\delta)\cdot|L_{MC}|\right) \geq \frac{1}{2}\left\lfloor \frac{1}{2\epsilon+W^{-1}} \right\rfloor + \frac{1}{2}\log(1-2\delta) = \Omega\left(\frac{1}{\epsilon}\right)$$

Once again, the case for a $\Omega(\log W)$ lower bound is based on Lemma 4 and follows from similar arguments. ◀

3.2 Upper Bound

We now present an algorithm for Basic-Counting that provides a $W\epsilon$-additive approximation $\widehat{C^W}$ for C^W over a binary stream with near-optimal memory. Denote $k \triangleq \frac{1}{2\epsilon}$. For simplicity, we assume that $\frac{W}{k}$ and k are integers. Intuitively, our algorithm partitions the stream into k *blocks* of size $\frac{W}{k}$, representing each using a single bit. A set bit corresponds to a count of $\frac{W}{k}$ in the input stream, while a clear bit corresponds to a count of 0. We then use an "optimistic" approach to reduce the error – the number of ones in the input stream not counted using the bit array is *propagated* to the next block; this means a block might be represented with 1, even if it contains only a single set bit. Surprisingly, we show that this approach allows us to keep the error bounded and that the errors do not accumulate. We keep a counter y for the number of 1s. At the end of a block, if y is larger than $\frac{W}{k}$, we mark the current block and subtract $\frac{W}{k}$ from y, propagating the remainder to the next block. Our algorithm answers queries by multiplying the number of marked blocks in the current window by $\frac{W}{k}$, making corrections to reduce the error. We maintain the following variables:
$\boldsymbol{y}$ – a counter for the number of ones.
$\boldsymbol{b}$ – a bit-array of size k.
$\boldsymbol{i}$ – the index of the "oldest" block in b.
$\boldsymbol{B}$ – the sum of all bits in b.
$\boldsymbol{m}$ – a counter for the current offset within the block.
Every arriving bit is handled as follows: We increment m, and if the bit is set we also increment y. At the end of a block, we check if y exceeds $\frac{W}{k}$. If so, we subtract $\frac{W}{k}$ from y and set the bit b_i. This way, the reduction in y is compensated for by the newly set bit in b. The previous value of b_i, holding information about 1s that just left the window, is forgotten.

To answer a query the algorithm returns the number of set bits in b multiplied by the block size $\frac{W}{k}$. We then add the value of y, which represents the number of ones not yet recorded in b, and subtract $m \cdot b_i$, as m bits of the oldest recorded block have already left the window. Finally, we remove any bias from the estimation by subtracting $\frac{W}{2k}$ (half a block).

In order to answer queries without iterating over b, we maintain another variable B, which keeps track of the number of ones in b. The entire pseudo-code is given in Algorithm 1.

▶ **Theorem 7.** *Algorithm 1 provides a $W\epsilon$-additive approximation of* Basic-Counting.

Proof. First, let us introduce some notations used in the proof. Assume that the index of the last bit is $W+m$, where x_W is the last bit of a block and $m < \frac{W}{k}$. b_i is considered after $W+m$ bits have been processed. We denote y_j the value of y *after* adding bit j.

The setting for the proof is given in Figure 1. We aim to approximate

$$C^W \triangleq \sum_{j=m+1}^{W+m} x_j. \tag{1}$$

Our algorithm uses the following approximation:

$$\widehat{C^W} = \frac{W}{k} \cdot B + y_{W+m} - \frac{W}{2k} - m \cdot b_i = \frac{W}{k} \cdot B + y_W + \sum_{j=W+1}^{W+m} x_j - \frac{W}{2k} - m \cdot b_i. \tag{2}$$

At times $1, 2, \ldots, W$, y is incremented once for every set bit in the input stream. At the end of block j, if y is reduced by $\frac{W}{k}$, then b_j is set and will not be cleared before time $W+m$. Therefore, $\frac{W}{k} \cdot B + y_W = y_0 + \sum_{j=1}^{W} x_j$. Substituting $\frac{W}{k} \cdot B + y_W$ in (2), we get

$$\widehat{C^W} = y_0 + \sum_{j=1}^{W} x_j + \sum_{j=W+1}^{W+m} x_j - \frac{W}{2k} - m \cdot b_i = y_0 + \sum_{j=1}^{m} x_j + \sum_{j=m+1}^{W+m} x_j - \frac{W}{2k} - m \cdot b_i.$$

Algorithm 1 Additive Approximation of Basic Counting

Initialization: $y = 0, b = 0, m = 0, i = 0$.
function ADD(Bit x)
  if $m = \frac{W}{k} - 1$ **then**
    $B = B - b_i$
    if $y + x \geq \frac{W}{k}$ **then**
      $b_i = 1$
      $y = y - \frac{W}{k} + x$
    else
      $b_i = 0$
      $y = y + x$
    $B = B + b_i$
    $m = 0$
    $i = i + 1 \mod k$
  else
    $y = y + x$
    $m = m + 1$
function QUERY
  return $\frac{W}{k} \cdot B + y - \frac{W}{2k} - m \cdot b_i$

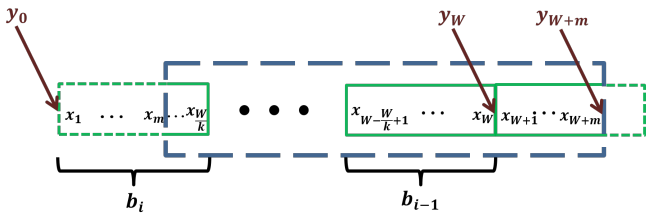

Figure 1 The setting for the proof of Theorem 7. b is cyclic – b_i represents the oldest block and b_{i-1} the newest completed block.

Plugging the definition of C^W, we get $\widehat{C^W} = y_0 + \sum_{j=1}^{m} x_j + C^W - \frac{W}{2k} - m \cdot b_i$. Therefore, the error is

$$\widehat{C^W} - C^W = y_0 + \sum_{j=1}^{m} x_j - m \cdot b_i - \frac{W}{2k}.$$

We consider two cases:

$b_i = 1$: This means that y had crossed the threshold by time $\frac{W}{k}$, i.e. $y_0 + \sum_{j=1}^{\frac{W}{k}} x_j \geq \frac{W}{k}$ and equivalently $y_0 + \sum_{j=1}^{m} x_j \geq \frac{W}{k} - \sum_{j=m+1}^{\frac{W}{k}} x_j$. Thus, on one side

$$\begin{aligned}\widehat{C^W} - C^W = y_0 + \sum_{j=1}^{m} x_j - m - \frac{W}{2k} &\geq \frac{W}{k} - \sum_{j=m+1}^{\frac{W}{k}} x_j - m - \frac{W}{2k} \\ &\geq \frac{W}{k} - \left(\sum_{j=m+1}^{\frac{W}{k}} 1\right) - m - \frac{W}{2k} \geq -\frac{W}{2k} = -W\epsilon.\end{aligned}$$

To bound the error from above we use the fact that the value of y at the end of a block never exceeds $\frac{W}{k}$. This can be shown by induction, as y is incremented at most $\frac{W}{k}$

times during one block, and then reduced by $\frac{W}{k}$ if it exceeds the block size. Therefore, $\widehat{C^W} - C^W = y_0 + \sum_{j=1}^{m} x_j - m - \frac{W}{2k} \leq y_0 - \frac{W}{2k} \leq \frac{W}{k} - \frac{W}{2k} = W\epsilon$.

$b_i = 0$: Similarly, this means that y was lower than the threshold at the end of block i, hence $y_0 + \sum_{j=1}^{\frac{W}{k}} x_j \leq \frac{W}{k} - 1$ or equivalently, $y_0 + \sum_{j=1}^{m} x_j \leq \frac{W}{k} - \sum_{j=m+1}^{\frac{W}{k}} x_j - 1$. Thus, our error is bounded from below by $\widehat{C^W} - C^W = y_0 + \sum_{j=1}^{m} x_j - \frac{W}{2k} \geq y_0 - \frac{W}{2k} \geq -\frac{W}{2k} = -W\epsilon$ and from above by

$$\widehat{C^W} - C^W = y_0 + \sum_{j=1}^{m} x_j - \frac{W}{2k} \leq \frac{W}{k} - \sum_{j=m+1}^{\frac{W}{k}} x_j - \frac{W}{2k} - 1 \leq \frac{W}{2k} - 1 \quad = W\epsilon - 1.$$

We have established that in all cases the absolute error is at most $W\epsilon$ as required. ◀

We next prove that the memory requirement of Algorithm 1 is nearly optimal.

▶ **Theorem 8.** *Algorithm 1 requires $\frac{1}{2\epsilon} + 2\log W + O(1)$ bits of memory.*

Proof. We represent y using $\lceil 2 + \log(W\epsilon) \rceil$ bits, m using $\lceil 1 + log(W\epsilon) \rceil$ bits and b using k bits. Additionally, i requires $\lceil \log k \rceil$ bits, and B another $\lceil \log(k+1) \rceil$ bits. Overall, the number of bits required is $k + \lceil 2 + \log(W\epsilon) \rceil + \lceil 1 + \log(W\epsilon) \rceil + \lceil \log k \rceil + \lceil \log(k+1) \rceil$ $\leq k + 2\log(W\epsilon) - 2\log(2\epsilon) + 8 = \frac{1}{2\epsilon} + 2\log(W) + 6 = \frac{1}{2\epsilon} + 2\log W + O(1)$. ◀

Theorem 5 shows that our algorithm uses at most twice as much memory as required by the lower bound (up to a constant number of bits) for every *constant* $\epsilon \leq \frac{1}{4}$. When ϵ is not constant, our memory requirement is at most 3 times the lower bound, as shown in the following lemma.

▶ **Corollary 9.** *For any $\epsilon \leq \frac{1}{4}$, the ratio between the memory consumption of Algorithm 1 and the lower bound for additive approximations for* Basic-Counting *is*

$$\frac{\frac{1}{2\epsilon} + 2\log W + O(1)}{\max\left\{\log W, \frac{1}{2\epsilon + W^{-1}}\right\}} = 3 + o(1).$$

Since the proof is very technical, and due to lack of space, it is left for the full version [4].

4 Basic-Summing Problem

We now consider an extension of Basic-Counting where elements are non-negative integers:

▶ **Definition 10** (Basic-Summing)**.** Given a stream of elements comprising of integers in the range $[R+1] = \{0, 1, \ldots, R\}$, maintain the sum S of the last W elements.

4.1 Lower Bound

We now show that approximating Basic-Summing to within an additive error of $RW\epsilon$ requires $\Omega(\frac{1}{\epsilon} + \log W)$ bits for $\epsilon \geq \frac{1}{2W}$ and $\Omega(W \log\left(\frac{1}{W\epsilon}\right))$ bits for $\epsilon \leq \frac{1}{2W}$.

▶ **Lemma 11.** *For any $\epsilon \leq \frac{1}{4}$, approximating* Basic-Summing *to within an additive error of $RW\epsilon$ requires $\left\lfloor \max\left\{\log W, \frac{1}{2\epsilon + W^{-1}}\right\} \right\rfloor$ memory bits.*

Proof. The proof of the lemma is very similar to the proof of Theorem 5 and is obtained by replacing every set bit with the integer R in Lemma 3 and Lemma 4. ◀

Next, we show a lower bound for smaller values of ϵ.

▶ **Lemma 12.** *For any ϵ, approximating* Basic-Summing *to within an additive error of $RW\epsilon$ requires at least $W \log \left\lfloor \frac{1}{4W\epsilon} + 1 \right\rfloor$ memory bits.*

Proof. Denote $x \triangleq \lfloor 2RW\epsilon + 1 \rfloor$ and $C \triangleq \left\{ n \cdot x \mid n \in \left\{ 0, 1, \ldots, \left\lfloor \frac{1}{2W\epsilon + R^{-1}} \right\rfloor \right\} \right\}$. Let L be the language of all W length strings over the number in C, i.e.,

$$L_{R,W,\epsilon} = \{\sigma_0\sigma_1 \cdots \sigma_{W-1} | \forall j \in [W] : \sigma_j \in C\}.$$

We show that every two distinct sequences in L must lead the algorithm into distinct configurations implying a lower bound of

$$\lceil \log |L| \rceil \geq W \log |C| = W \log \left\lfloor \frac{1}{2W\epsilon + R^{-1}} + 1 \right\rfloor \geq W \log \left\lfloor \frac{1}{4W\epsilon} + 1 \right\rfloor$$

bits, where the last inequality follows from the fact that any $\epsilon < \frac{1}{2RW}$ implies exact summing. Assume, by way of contradiction, that two different words

$$s^1 = \sigma_0^1\sigma_1^1 \cdots \sigma_{W-1}^1, s^2 = \sigma_0^2\sigma_1^2 \cdots \sigma_{W-1}^2 \in L$$

lead the algorithm to the same configuration. Denote the index of the last letter that differs between s^1 and s^2 by $t \triangleq max\{\tau \mid \sigma_\tau^1 \neq \sigma_\tau^2\}$. Next, consider the sequences $s^1 \cdot 0^{t-1}$ and $s^2 \cdot 0^{t-1}$. The algorithm must reach the same configuration after processing these sequences, even though the sum of the last W elements differ by at least $x = \lfloor 2RW\epsilon + 1 \rfloor > 2RW\epsilon$. Therefore, the algorithm's error must be greater than $RW\epsilon$ at least for one of the sequences, in contradiction to the assumption. ◀

▶ **Theorem 13.** *Approximating* Basic-Summing *to within an additive error of $RW\epsilon$ requires $\Omega(\frac{1}{\epsilon} + \log W)$ bits for $\frac{1}{2W} \leq \epsilon \leq \frac{1}{4}$ and $\Omega(W \log\left(\frac{1}{W\epsilon}\right))$ bits for $\epsilon \leq \frac{1}{2W}$.*

Proof. Lemma 11 shows that approximating Basic-Summing within $RW\epsilon$ requires

$$\max\left\{\log W, \frac{1}{2\epsilon + W^{-1}}\right\}$$

bits for $\frac{1}{2W} \leq \epsilon \leq \frac{1}{4}$. The same argument used in Lemma 9 shows that this implies $\Omega\left(\frac{1}{\epsilon} + \log W\right)$ bits lower bound for any $\epsilon \geq \frac{1}{2W}$. For $\epsilon < \frac{1}{2W}$ such that $\epsilon = \Theta(W^{-1})$, approximating Basic-Summing within $RW\epsilon$ implies a $\frac{R}{2}$-additive approximation and therefore the $\Omega\left(\frac{1}{\epsilon} + \log W\right)$ bound holds. For $\epsilon = o\left(\frac{1}{W}\right)$, we use Lemma 12, which implies a lower bound of $W \log \left\lfloor \frac{1}{4W\epsilon} + 1 \right\rfloor = \Omega\left(W \log\left(\frac{1}{W\epsilon}\right)\right)$ memory bits. ◀

An immediate corollary of Theorem 13 is that any exact algorithm for Basic-Summing requires at least $\Omega(W \log R)$ bits, i.e., the naive solution of maintaining a W-sized array of the elements in the window, encoding each using $\lceil \log(R+1) \rceil$ bits, is optimal (for exact Basic-Summing). Finally, we extend the results to randomized algorithms, where the proof is left for the full version [4] due to lack of space.

▶ **Theorem 14.** *For any fixed $\delta \in [0, 1/2)$, any randomized Monte Carlo algorithm that gives a $W\epsilon$ approximation to* Basic-Summing *with a probability of at least $1 - \delta$ requires $\Omega(\frac{1}{\epsilon} + \log W)$ bits for $\frac{1}{2W} \leq \epsilon \leq \frac{1}{4}$ and $\Omega(W \log\left(\frac{1}{W\epsilon}\right))$ bits for $\epsilon \leq \frac{1}{2W}$. Notice that the $\delta = 0$ case applies to Las Vegas algorithms.*

4.2 Upper Bound

We show that our Basic-Counting algorithm can be adapted to this problem with only a small memory overhead such that the algorithm's state size remains independent of R. We first present the extension of the algorithm for the $\epsilon^{-1} \leq 2W\left(1 - \frac{1}{\log W}\right)$ case. In Section 4.3 we complete the picture by giving an alternative algorithm for smaller values of ϵ. Intuitively, we "scale" the algorithm by dividing each added element by R and rounding the result. In order to keep the sum of elements not yet accounted for in b, y is now maintained as a fixed-point variable rather than an integer. Ideally, the fractional value of the remainder y should allow exact representation of $\{0, 1/R, \ldots, 1 - 1/R\}$, and therefore requires $\log R$ bits. When the range R is "large", or more precisely $R = \omega(\epsilon^{-1})$, we save space by storing the fractional value of y using less than $\log R$ bits, which inflicts a rounding error. That is, we keep y using $\left\lceil \log\left(2\frac{W}{k}\right)\right\rceil + \upsilon$ bits. Similarly to our Basic-Counting algorithm, $\left\lceil \log\left(2\frac{W}{k}\right)\right\rceil$ bits are used to store the integral part of y. The additional υ bits are used for the fractional value of y. The value of υ is determined later.

In order to keep the total error bounded, we compensate for the rounding error by using smaller block sizes, which are derived from the number of blocks k, determined in (3). Our algorithm keeps the following variables:

$\boldsymbol{b}$ – a bit-array of size k.

$\boldsymbol{y}$ – a counter for the sum of elements which is not yet accounted for in b.

$\boldsymbol{i}$ – the index of the "oldest" block in b.

$\boldsymbol{B}$ – the sum of all bits in b.

$\boldsymbol{m}$ – a counter for the current offset within the block.

Our Basic-Summing algorithm is presented in Algorithm 2. We use $\text{Round}_\upsilon(z)$ for some $z \in [0, 1]$ to denote rounding of z to the nearest value $\widetilde{z}$ such that $2^\upsilon \widetilde{z}$ is an integer.

Algorithm 2 Additive Approximation for Basic-Summing

```
1: Initialization: y = 0, b = 0, B = 0, i = 0, m = 0.
2: function ADD(ELEMENT x)
3:     x' = Round_υ(x/R)
4:     if m = W/k − 1 then
5:         B = B − b_i
6:         if y + x' ≥ W/k then
7:             b_i = 1
8:             y = y − W/k + x'
9:         else
10:            b_i = 0
11:            y = y + x'
12:        B = B + b_i
13:        m = 0
14:        i = i + 1 mod k
15:    else
16:        y = y + x'
17:        m = m + 1
18: function QUERY()
19:    return R · (W/k · B + y − W/2k − m · b_i)
```

Algorithm 3 Additive Approximation for Basic-Summing with Small Error

```
1: Initialization: y = 0, b = 0, B = 0, i = 0.
2: function ADD(element x)
3:     x' = Round_υ(x/R)
4:     B = B − b_i
5:     b_i = ⌊(y + x')/(W/k)⌋
6:     y = y + x' − b_i · W/k
7:     B = B + b_i
8:     i = i + 1 mod W
9: function QUERY()
10:    return R · (W/k · B + y − W/(2k))
```

▶ **Theorem 15.** *For any* $\epsilon^{-1} \leq 2W\left(1-\frac{1}{\log W}\right)$*, Algorithm 2 provides an* $RW\epsilon$*-additive approximation for* Basic-Summing.

Theorem 15 shows that for any $\epsilon^{-1} \leq 2W\left(1-\frac{1}{\log W}\right)$, by choosing $\upsilon \triangleq \left\lceil \log\left(\epsilon^{-1}\log W\right)\right\rceil$ and the number of blocks to be

$$k \triangleq \left\lceil \frac{1}{2\epsilon - 2^{-\upsilon}} \right\rceil, \tag{3}$$

our algorithm estimates S with an additive error of $RW\epsilon$. Due to lack of space, the proof of Theorem 15 can be found in the full version [4]. The following theorem analyzes the memory requirements of our algorithm.

▶ **Theorem 16.** *For any* $\epsilon^{-1} \leq 2W\left(1-\frac{1}{\log W}\right)$*, Algorithm 2 requires* $\left(2\log W + \frac{1}{2\epsilon}\right)(1+o(1))$ *memory bits.*

The proof is similar to the proof of Theorem 8 and therefore appears in the full version [4].

4.3 Summing with Small Error

Algorithm 2 only works for $\epsilon^{-1} \leq 2W\left(1-\frac{1}{\log W}\right)$ that satisfies $\frac{W}{k} \geq 1$; otherwise, k cannot represent the number of blocks, as blocks cannot be empty. To complete the picture, we present Algorithm 3 that works for smaller errors. Intuitively, we keep an array b of size W, such that every cell represents the number of integer multiples of $\frac{RW}{k}$ in an arriving item. Similarly to the above algorithms, we reduce the error by tracking the remainder in a variable y, propagating uncounted fractions to the following item. In this case as well, the optimistic approach reduces the error compared with keeping a W-sized array of rounded values for approximating the sum. Each cell in b needs to represent a value in $\left\{0,1,\ldots,\left\lfloor 1+\frac{k}{W}\right\rfloor\right\}$; the remainder y is now a fractional number, represented using υ bits. When a new item is added, we scale it, add the result to y, and update both b_i and the remainder.

▶ **Theorem 17.** *Algorithm 3 provides an* $RW\epsilon$*-additive approximation for* Basic-Summing.

The proof appears in the full version [4]. It considers the rounding error generated by representing x' using υ bits, and shows that the remainder propagation (Line 6) limits error accumulation.

▶ **Theorem 18.** *For any* $\epsilon^{-1} > 2W\left(1-\frac{1}{\log W}\right) = 2W(1-o(1))$*, Algorithm 3 requires* $W\log\left(\frac{1}{2W\epsilon}+1\right)\cdot(1+o(1)) \leq \frac{1}{2\epsilon}\cdot(1+o(1))$ *memory bits.*

The proof is similar to the proof of Theorem 8 and therefore appears in the full version [4].

We conclude the section by showing that our algorithm is succinct, requiring only $(1+o(1))$ times as much memory as the lower bound proved in Theorem 13.

▶ **Theorem 19.** *Let $\epsilon = o(W^{-1})$, and denote $\mathcal{B} \triangleq W \log \left\lfloor \frac{1}{4W\epsilon} + 1 \right\rfloor$. Algorithm 3 provides $RW\epsilon$ additive approximation to* Basic-Summing *using $\mathcal{B} \cdot (1 + o(1))$ memory bits.*

Proof. Theorem 18 shows that the number of bits our algorithm requires for $\epsilon = o(W^{-1})$ is $W \log \left(\frac{1}{2W\epsilon} + 1\right) \cdot (1 + o(1)) \leq \mathcal{B}(1 + \frac{2W}{\mathcal{B}}) \cdot (1 + o(1)) = \mathcal{B} \cdot (1 + o(1))$. ◀

5 Discussion

In this paper, we have investigated additive approximations for the Basic-Counting and Basic-Summing problems. For both cases, we have provided space efficient algorithms. Further, we have proved the first lower bound for additive approximations for the Basic-Counting problem, and showed that our algorithm achieves this bound, and is hence optimal. In the case of Basic-Summing, whenever $\epsilon^{-1} \leq 2W\left(1 - \frac{1}{\log W}\right)$, the same lower bound as in the Basic-Counting problems still holds and so our approximation algorithm for this domain is optimal up to a small factor. For other values of ϵ, we have shown an improved lower bound and a corresponding succinct approximation algorithm.

In the future, we would like to study lower and upper bounds for additive approximations for several related problems. These include, e.g., approximating the sliding window sum of weights for each item in a stream of (item, weight) tuples. Further, we intend to explore applying additive approximations in the case of multiple streams. Obviously, one can allocate a separate counter for each stream, thereby multiplying the space complexity by the number of concurrent streams. However, it was shown in [13] that for the case of multiplicative approximations, there is a more space efficient solution. We hope to show a similar result for additive approximations.

Acknowledgments. We thank Dror Rawitz for helpful comments. This work was partially funded by MOST grant #3-10886 and the Technion-HPI research school.

References

1 Michael H Albert, Alexander Golynski, Angèle M Hamel, Alejandro López-Ortiz, S Srinivasa Rao, and Mohammad Ali Safari. Longest increasing subsequences in sliding windows. *Theoretical Computer Science*, 321(2):405–414, 2004.

2 Arvind Arasu and Gurmeet Singh Manku. Approximate counts and quantiles over sliding windows. In *Proc. of the 23rd ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, PODS 2004*. Association for Computing Machinery, Inc., June 2004.

3 Brian Babcock, Mayur Datar, Rajeev Motwani, and Liadan O'Callaghan. Maintaining variance and k-medians over data stream windows. In Frank Neven, Catriel Beeri, and Tova Milo, editors, *Proceedings of the Twenty-Second ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 9-12, 2003, San Diego, CA, USA*, pages 234–243. ACM, 2003.

4 Ran Ben Basat, Gil Einziger, Roy Friedman, and Yaron Kassner. Efficient summing over sliding windows. *CoRR*, abs/1604.02450, 2016. URL: `http://arxiv.org/abs/1604.02450`.

5 Ran Ben Basat, Gil Einziger, Roy Friedman, and Yaron Kassner. Heavy hitters in streams and sliding windows. In *INFOCOM, 2016 Proceedings IEEE*, pages 307–315, April 2016.

6 Vladimir Braverman, Ran Gelles, and Rafail Ostrovsky. How to catch l2-heavy-hitters on sliding windows. *Theoretical Computer Science*, 554:82–94, 2014.

7 Vladimir Braverman and Rafail Ostrovsky. Smooth histograms for sliding windows. In *Foundations of Computer Science, 2007. FOCS'07. 48th Annual IEEE Symposium on*, pages 283–293. IEEE, 2007.

8 Edith Cohen and Martin J. Strauss. Maintaining time-decaying stream aggregates. *J. Algorithms*, 59(1):19–36, 2006.

9 Graham Cormode and Ke Yi. Tracking distributed aggregates over time-based sliding windows. In *Scientific and Statistical Database Management*, pages 416–430. Springer, 2012.

10 Michael Crouch and Daniel S. Stubbs. Improved streaming algorithms for weighted matching, via unweighted matching. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2014, September 4-6, 2014, Barcelona, Spain*, pages 96–104, 2014. doi:10.4230/LIPIcs.APPROX-RANDOM.2014.96.

11 Michael S Crouch, Andrew McGregor, and Daniel Stubbs. Dynamic graphs in the sliding-window model. In *Algorithms–ESA 2013*, pages 337–348. Springer, 2013.

12 Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Maintaining stream statistics over sliding windows. *SIAM J. Comput.*, 31(6):1794–1813, 2002.

13 Phillip B. Gibbons and Srikanta Tirthapura. Distributed streams algorithms for sliding windows. In *SPAA*, pages 63–72, 2002.

14 Regant Y.S. Hung and H. F. Ting. Finding heavy hitters over the sliding window of a weighted data stream. In E. Laber, C. Bornstein, L. Nogueira, and L. Faria, editors, *LATIN 2008: Theoretical Informatics*, volume 4957 of *LNCS*, pages 699–710. Springer, 2008. doi:10.1007/978-3-540-78773-0_60.

15 Lap-Kei Lee and H. F. Ting. Maintaining significant stream statistics over sliding windows. In *Proceedings of the Seventeenth Annual Symposium on Discrete Algorithms, SODA*, pages 724–732. ACM Press, 2006.

16 Lap-Kei Lee and HF Ting. A simpler and more efficient deterministic scheme for finding frequent items over sliding windows. In *Proc. of the SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 290–297. ACM, 2006.

17 Yang Liu, Wenji Chen, and Yong Guan. Near-optimal approximate membership query over time-decaying windows. In *INFOCOM, Proceedings IEEE*, pages 1447–1455, April 2013.

18 Kyriakos Mouratidis, Spiridon Bakiras, and Dimitris Papadias. Continuous monitoring of top-k queries over sliding windows. In *Proc. of the International Conference on Management of Data*, SIGMOD, pages 635–646, New York, NY, USA, 2006. ACM.

19 Moni Naor and Eylon Yogev. Sliding bloom filters. In Leizhen Cai, Siu-Wing Cheng, and Tak-Wah Lam, editors, *Algorithms and Computation*, volume 8283 of *Lecture Notes in Computer Science*, pages 513–523. Springer Berlin Heidelberg, 2013. doi:10.1007/978-3-642-45030-3_48.

20 Krešimir Pripužić, Ivana Podnar Žarko, and Karl Aberer. Time- and space-efficient sliding window top-k query processing. *ACM Trans. Database Syst.*, 40(1):1:1–1:44, March 2015.

21 Hong Shen and Yu Zhang. Improved approximate detection of duplicates for data streams over sliding windows. *Journal of Computer Science and Technology*, 23(6):973–987, 2008.

22 Zhitao Shen, M.A. Cheema, Xuemin Lin, Wenjie Zhang, and Haixun Wang. Efficiently monitoring top-k pairs over sliding windows. In *Data Engineering (ICDE), 2012 IEEE 28th International Conference on*, pages 798–809, April 2012.

23 Andrew Chi-Chin Yao. Probabilistic computations: Toward a unified measure of complexity. In *18th Annual Symp. on Foundations of Computer Science*, pages 222–227. IEEE, 1977.

24 Wenjie Zhang, Ying Zhang, Muhammad Aamir Cheema, and Xuemin Lin. Counting distinct objects over sliding windows. In *Proceedings of the Twenty-First Australasian Conference on Database Technologies – Volume 104*, ADC'10, pages 75–84, Darlinghurst, Australia, Australia, 2010. Australian Computer Society, Inc.