Second Frequency Moments on Sliding Windows

Aidan Fowler, Greg Vorsanger, David Widen

December 9, 2013

1 Introduction

In their seminal 1996 work, Alon, Matias, and Szegedy evaluated the ability to calculate frequency moments in the streaming model. Since this work, research in streaming algorithms has expanded dramatically, and many new streaming models have been introduced. One such model is the Sliding Window Model, in which elements are given timestamps and older elements expire as time progresses. We found this model interesting as it both introduces additional technical challenges in terms of theory and describes an important practical observation; sometimes old data becomes less important.

In general, streaming algorithms compute some statistic on a large data set, a data stream, with a constant number of passes over the data, typically only a single pass. It is assumed that the data stream being processed is significantly larger than main memory or even disk space, and that there is limited time to process each element. These constraints lead many streaming algorithms to be approximation algorithms that can compute an accurate result to within some error of the optimal solution. More specifically:

Let m and n be positive integers. A **stream** D = D(n,m) is a sequence of integers $p_1, \ldots, p_m$, where $p_i \in \{1, \ldots, n\}$. A **frequency vector** is a vector of dimensionality n with non-negative entries $f_i, i \in [n]$ defined as:

$$f_i = |\{j : 1 \leq j \leq m, p_j = i\}|$$

The Sliding Window Model is a variation on the generic streaming model. At any point within the stream, there is a limited continuous subset of stream elements that are considered active, called the window. As a new element enters the window, the oldest element of the window expires. The computation will focus on the window and ignore all other elements that have expired. More specifically:

The **Sliding Window Model** is a streaming model. Given stream D, every element $p_i \in D$ is given a timestamp. At any time define substream D_L, which consists of the most recent L elements. All computations are performed on the substream D_L.

For our final project we chose to focus on one computation in the Sliding Window Model, the second frequency moment. The second frequency moment is a method of measuring the occurences of individual elements in a data stream, and weighs more frequent elements more heavily when attempting to provide a single statistic describing the stream. This is a commonly studied

function, and an optimal algorithm for solving this in the general streaming model was provided in the work of Alon, Matias, and Szegedy. While algorithms that are optimal for certain groups of parameters have been provided in the streaming model, an approximation algorithm optimized for all parameters has not yet been proven.

2 Why the problem is important

Generally, the rise in importance of streaming algorithms is a direct consequence of an increase in the amount of data being collected and the desire for difficult computations to be performed upon these massive datasets. Intuitively, as these data sets get larger and larger, making multiple passes becomes infesible from a time perspective, and the cost of creating computers with memory that is of linear size to the large datasets being computed is practically impossible. Instead, new, sublinear memory solutions must be created using new analysis techniques in order to meet this computational demand.

In practice, the Sliding Window Model will be beneficial when there is a large amount of high throughput data to analyze, where older data is less revelant. We will discuss several different situations when this will occur.

One example of such an application is a city during a state of emergency, perhaps a natural disaster. In this event, many people will be making calls to 911 and there will not be enough operators to process all of these calls. There will be many more people that need help than police officers or national guardsmen who will be able to respond to these calls one-by-one. In this case, the set of data analyzed is information about the phone calls that come into the 911 dispatcher, specifically their location within the city and the time of the call. We also assume that on average, there is a bias towards more recent events to handle newer emergencies that have not been addressed yet. A one pass algorithm can place police officers and national guardsmen in locations to maximize their efficiency of helping people in need by treating the stream of 911 calls and breaking it into an active and inactive section: active calls will be 911 calls made within the average response time of ten minutes. This would allow disaster relief to put officers and guardsmen in locations where the most calls are coming from, and move them around based on when calls are handled and completed, thus increasing the effective aid given to the people.

Consider some ISP that needs to dynamically scale up or scale down network performance to a set of users to minimize the cost of providing service while maintaining their advertised bandwidth. All the ISP is able to do is monitor packets, and they know that the one million most recent sent and received packets are relevant. Any packets sent before that will have no tangible relationship to the amount of bandwidth their users currently need. The ISP can take their data stream and their window will be the one million most recent packets sent. The ISP can then compute the average throughput each host requires based on all of the packets being sent to either scale up or scale down the network performance to meet these needs. They can even do this with a block of blocks of users to determine the average bandwidth that each block of users requires, and then scale up or down each blocks performance based on the previous algorithm. Some national security agency could be interested in detecting short Distributed Denial of Service (DDoS) attacks on government or corporate machines. They could monitor all traffic that traverses the adjacent nodes to their

target webserver. Their sliding window could be the million most recent packets sent across the network. Their algorithm could detect the early stages of a DDoS attack by comparing network information in the current window to some long-term average for that specific server. This could be an early warning system to minimize damage done by such attacks, or even be used to launch counter-attacks.

The Sliding Window Model can also be used to detect the spin up of Amazon Elastic Compute Cloud (EC2) instances that could be used as part of some DDoS or Botnet Attack. Consider our stream as packets being sent in and out of one of Amazons EC2 regions. Our window could monitor the X most recent messages sent. Each message will have a different type, for example GET, POST, etc. Some of these messages will be POSTs with the purpose of spinning up machines. An algorithm can monitor the frequency of these spin up commands to determine if a large set of compute resources are about to be used, presumably for some attack.

3 Definitions

We define the following terms to be used to describe the work we will be analyzing:

Definition 3.1. *Let m and n be positive integers. A **stream** D = D(n,m) is a sequence of integers $p_1, \ldots, p_m$, where $p_i \in \{1, \ldots, n\}$. A **frequency vector** is a vector of dimensionality n with non-negative entries $f_i, i \in [n]$ defined as:*

$$f_i = |\{j : 1 \leq j \leq m, p_j = i\}|$$

Definition 3.2. *A k-th **frequency moment** of a stream D is defined by $F_k(D) = \sum_{i \in [n]} f_i^k$. Also, $F_\infty = \max_{i \in [n]} f_i$.*

Definition 3.3. *A **second frequency moment** the frequency moment F_k where $k = 2$.*

Definition 3.4. *The **Sliding Window Model** is a streaming model. Given stream D, every element $p_i \in D$ is given a timestamp. At any time define substream D_L, which consists of the most recent L elements. All computations are performed on the substream D_L.*

4 Background

The field of streaming algorithms has grown vastly over the last 15 years, with many lines of research focusing on many different models.

In their breakthrough work, Alon Matias and Szegedy observed a major difference in the space complexity of computing solutions for large frequency moments, F_k s.t $k > 2$ and small frequency moments, $k \leq 2$. In fact, while large frequency moments required sublinear space, small frequency moments only required polylogarithmic space.

As the focus of this project on exploring the path of one of the problems covered in the work of Alon, Matias, and Szegedy, despite many of the other interesting breakthroughs that may have

influenced the works we discuss, the background will now focus on specific results that relate to our problem.

The goal of our project is to analyze and hopefully improve the work of Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani from their 2002 paper *Maintaining Stream Statistics over Sliding Windows*. This work provides the major breakthroughs in moving computations from the general streaming model to the sliding window model. In this paper, they provide impressive algorithms using a data structure known as an exponential histogram. In this data structure they divisiide the stream into "buckets" of constantly increasing size.

While their paper provides analysis for multiple computations, we intend to focus on there solution of the L_2 norm, and consequently the second frequency moment. Additionally, we intend to comment on the work presented by Braverman and Ostrovsky in *Effective Computations on Sliding Windows*, which provides an improved bound on the second frequency moment using a technique known as smooth histograms.

5 Basic Counting

We now describe the Basic Counting Algorithm of Datar et al.

Definition 5.1. The Basic Counting Problem: *Given a stream of data elements, consisting of 0's and 1's, maintain at every time instant the count of the number of 1's in the last N elements.*

NOTE: The space complexity of this problem is measured in terms of the number of bits of memory required to store all information used to compute the solution.

The exact solution to this problem requires $\theta(n)$ bits of memory. It is prohibitive to use $\theta(n)$ bits of memory in most applications. For example, if we are trying to count data packets passed through a router, there will be far too many packets passed in any time frame that is not very small to record each one.

There is a good approximation for the basic counting problem that uses $o(n)$ memory.

Naive Solutions: Why They Do Not Suffice:
1. One naive solution to this problem is to maintain a counter which gets incremented upon the arrival of each element that is a 1. At each time instant one old data element expires but there is no way of knowing if the most recent expired element was a 1 or a 0. This solution does not work because we need to know this in order to decrement the counter.
2. Another naive solution that could be considered is using random sampling to maintain a sample of the window elements. This solution fails when the number of 1's is small.
3. The last naive solution would be to maintain histograms of the elements in the window. These histogram techniques require interpolation inside buckets to estimate the answer as some elements in a bucket may have expired. One histogram technique is to divide the window into k equiwidth buckets. The problem with this solution is the distribution of the number of 1's in buckets may be non-uniform. This will cause the error to be large when we interpolate inside a bucket that holds a

majority of the 1's in the window. If we use buckets of non uniform width in order to hold a close to uniform number of 1's in each bucket, we still run into errors because as time progresses, the uniformity of the number of 1's in each bucket can not be assured.

Solution:
In this section we describe a solution to the basic counting problem which uses $O(\frac{1}{\epsilon}log^2(N))$ bits of memory and provides an estimate within a $(1+\epsilon)$ factor of the exact answer. The solution works with $O(log^2N)$ bits of memory as long as the window size is bounded by N. The algorithm takes $O(1)$ amortized time per arrival of a new element, the worst case is $O(logN)$ time.

Algorithm: In order to solve the basic counting problem, we can maintan a histogram that records the timestamp of 1's that are "active" (belong in the silding window containing the last N elements). Elements arrive from the right, elements on the left are ones that are already seen. Each element has an arrival time that gets incremented by 1 for each arrival. Elements also have a timestamp which corresponds to the position of active data elements. The most recent elements get a timestamp of 1. Since we do not want to make updates to the timestamp at each arrival, we can use a wraparound counter of $log(N)$ bits allowing us to extract the timestamp by comparison with the counter value. The notation is illustrated below.

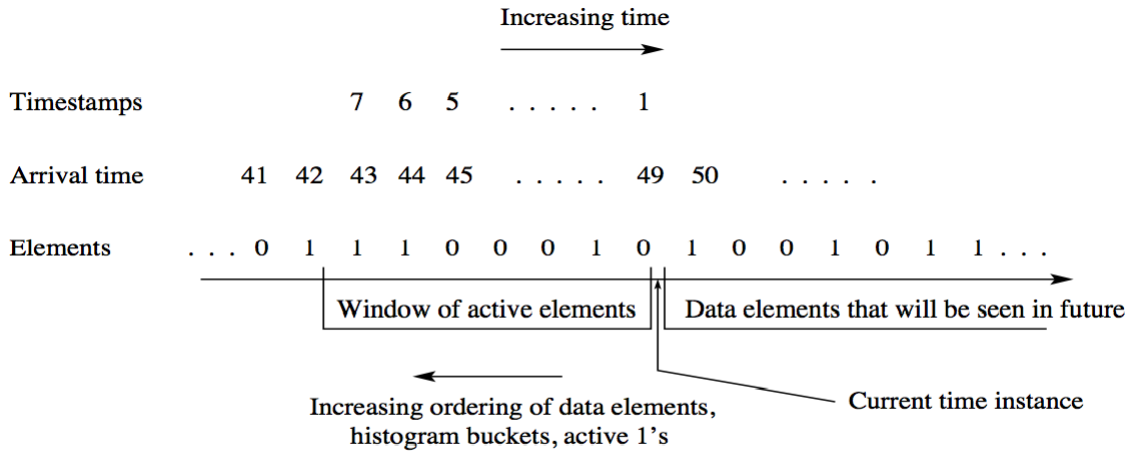

FIG. 1. *An illustration for the notation and conventions followed.*

In order to count the number of 1's, we maintain histograms of the active 1's in the data stream. Each bucket contains the timestamp of the most recent 1, and the number of 1's (bucket size). The timestamp of a bucket increases as new elements arrive. When a timestamp reaches N+1, the bucket has expired. There are no longer any active elements in it and we can drop the bucket and take back its memory. At any instant there is at most 1 bucket containing any 1's that may have expired. We can estimate the number of 1's by adding the bucket size of each bucket up to the last bucket and adding half the size of the last bucket (because the last bucket can contain anywhere from 1 to the bucket size number of active 1's). We estimate that this number of active 1's is half

the size of the bucket. Therefore, the absolute error is at most half the size of the last bucket.

Approximation Scheme:
Here we describe the technique to maintain buckets in order to guarantee the count error is at most ϵ. First, define $k = \frac{1}{\epsilon}$ and assume $\frac{k}{2}$ is an integer. As described above, the error in the estimate is $\frac{bucketsize}{2}$. The buckets are numbered from right to left, the most recent bucket is 1. We know that the count of active elements is at least $1 + \sum_{i=1}^{m-1} (BucketSize)_i$. The relative error is at most $(\frac{(BucketSize)_m}{2})(1 + \sum_{i=1}^{m-1} (BucketSize)_i)$ We ensure the error is at most $\frac{1}{k}$ by maintaining the following invariants:

Invariant 1: At all times the bucket sizes $C_1, ..., C_m$ are such that for all $j \leq m$, we have $(\frac{C_j}{2})(1 + \sum_{i=1}^{j-1} C_i) \leq \frac{1}{k}$
At any time let $N' \leq N$ be the number of active 1's. In order to use as few buckets as possible, we maintain buckets with exponentially increasing sizes following this second invariant.

Invariant 2: At all times the bucket size are non decreasing. Also the bucket sizes are constrained to the following: $1, 2, 4, ..., n^{m'}$ for some $m' \leq m$ and $m' \leq log\frac{2N}{k} + 1$. For every bucket size other than the last bucket, there are at most $\frac{k}{2} + 1$ and at least $\frac{k}{2}$ buckets of that size.

If invariant 2 is satisfied, we are guaranteed that there are at least $\frac{k}{2}$ buckets of sizes $1, 2, ..., 2^{r-1}$ which have indices less that j.
This implies $C_j \leq (\frac{2}{k})(1 + \sum_{i=1}^{j-1} C_i)$ If invariant 2 is satisfied, invariant 1 is also satisfied. In order to cover all the active 1's we need no more than $m \leq (\frac{k}{2} + 1)(log(\frac{2k}{k} + 1) + 1)$ buckets. Each bucket has a size and a timestamp. The bucket size takes at most $log(n)$ values, we can maintain them using $log(log(n))bits$ The timestamp requires $log(n)$ bits. The total memory requirement is $log(n) + log(log(n))$ bits. The total memory for the historgram is $O(\frac{1}{\epsilon} log^2(N))$

Merging two buckets corresponds to creating a new bucket whose size is equal to the sum of the sizes of the two buckets and whose timestamp is the timestamp of the most recent bucket. Cascading may require $\theta(log(\frac{2n}{k}))$ mergers. The amortized cost of mergers is $O(1)$. If instead of maintaining a timestamp for every bucket we maintain a timestamp for the most recent bucket and store the difference between the timestamps for consecutive buckets, we can reduce the total memory to $O(klog^2(\frac{N}{k}))$.

5.1 Beyond 0's and 1's

The authors then provide a description to expand the algorithm to any range of numbers for input. They describe this problem as the sum problem, as the computation is effectively the sum of all numbers inside of the window.

Simply, the authors recognize that by allowing for non-unique timestamps, the basic counting algorithm is only expanded slightly, with the upper bound being increased largely by the range of elements being computed. The basic principle is that to represent the number "50" being taken as input, they treat the problem as an instance of basic counting where fifty instances of the number "1" arrive at one timestamp. This technique allows the algorithm to effectively follow similarly to basic counting, while working for any range of numbers R.

5.2 Basic Counting as a Method for Calculating Frequency Moments

In addition to the basic counting algorithm and sum algorithm, the authors provide an expansion of the underlying data structure, the exponential histogram, to also analyze other functions that hold under a set of restrictions. They then prove that the L_p norms are one such group of function, which is computationally identical to F_k. From this point, the authors provide a specific bound for $k \in 1, 2$.

5.3 Smooth Histograms and the Current Best Bound

Similarly to the work of Datar et al., Braverman and Ostrovsky provide another data structure, smooth histograms, that uses buckets and weakly additive functions to approximate functions on sliding windows.

6 Observations

6.1 General Notes About Sliding Windows

The work of Datar et al. provides an interesting insight into the differences in difficulty between different comptuational models. One of the major contributions the paper provides is an algorithm that allows for obtaining an accurate sum of numbers in a sliding window model. In the general streaming model, this problem is trivial, and for a stream of length M and elements ranging from $\{0, N\}$ only requires $\log(NM)$ space to compute the correct answer deterministically! This is a major difference in computational difficulty, and highlights the difference between the models. However, other functions, such as F_k, the computational difficulty is pretty similar for both models.

6.2 Weakly Additive Functions

Both exponential and smooth histograms rely upon the fact that functions are weakly additive. That is, $f(a) + f(b) = C * (f(a) + f(b))$ for some constant C. For the second Frequency moment, we show this C is at most 2. We show this in an attempt to possibly identify analysis based on C

that can be improved. The concept we had was that because analysis for L_P norms and frequency moments is frequently done generically for any k, we were interested in seeing if by fixing k we could improve the analysis and find a better bound. We were not able to find any case where this occured.

Lemma 6.1. *The maximum value of the addition of two values of F_2 for two vectors with known second frequency moments $F_{2,a}, F_{2,b}$ but unknown frequency vectors A, B is at most $2(F_{2,a} + F_{2,b})$*

Proof. Let $F_{2,ab}$ be the F_2 computed on the sum of frequency vectors A and B. Assume all elements are the same element in each of the two vectors. Thus, $F_a = F_b = |A|^2 = |B|^2$ The value of $F_{2,ab} = (a+b)^2 = a^2 + b^2 + 2ab$. However, a = b, therefore $F_{2,ab} = (a+b)^2 = 4a^2$. This is the maximum growth the function can obtain, therefore by simply adding $F_{2,a} + F_{2,b}$ we obtain at least 1/2 $F_{2,ab}$ □

7 Upper Bounds

For the sake of analysis, we will now attempt to manipulate the bounds of Datar et. al and Braverman and Ostrovsky to put them into forms where they are comparable. In general, we try to arrive in a notation similar to the work of Alon, Matias, and Szegedy. The work of Braverman and Ostrovsky is closer to this, but Datar et.al provide a more complex error bound.

We assume the the probability of success,δ, is non constant. Additonally, we do not hold ϵ to be a constant. Thus the following upper bound is obtained:

For a $(1+\epsilon)^2 * \frac{4}{k} + 1 + \epsilon$ approximation, we require $O(\log N(\log N + \log m = M) = O(\frac{1}{\epsilon^2}\log^2(M)\log(\frac{1}{\delta}$ space. This is roughly $O(\frac{1}{\epsilon^2}\log^3 M)$ space.

However, converting this to a rough $(1+\epsilon)$ approximation gives us a slighly different bound.

Lemma 7.1. *A $(1+\epsilon)$ approximation of F_2 takes at least $O(\frac{1}{\epsilon^4}\log^3(N)$ Space.*

Proof. For all non-constant ϵ:

$(1+\epsilon)^2 * \frac{4}{k} + 1 + \epsilon = 2(1+\epsilon)^2 + (1+\epsilon)$ for $k = 2$.

$2(1+\epsilon)^2 + (1+\epsilon) = 2\epsilon^2 + (3+\epsilon)$.

This error implies that in order to accurately create a $(1+\epsilon)$ bound, we would in fact need more than a quadradtic increase in the number of bits to achieve desired epsilon. □

The work of Braverman and Ostrovsky provide their bound in terms of a $(1+\epsilon)$ approximation: $O(\frac{1}{\epsilon^3}\log^2(N)\log\log(\frac{N}{\epsilon\delta})$. Thus, for non-constant ϵ and δ this bound provides improvement.

However, in the case that ϵ and δ are constant, the work of Datar et al. is the best bound with an asymptotic upper bound of O($\log^2 N$). This bound also matches the lower bound that they also provide in their paper, as described in the next section.

We found this to be an interesting point of analysis, especially as Datar et al. do not go into detail about their decisions as to why they choose to hold these parameters constant. We feel this is a good reminder as to why you always have to review the claims of academic work. While they provide an algorithm with an optimal bound, it doesn't mean a better algorithm cannot exist!

8 A Brief Discussion of Lower Bounds

In their work, Datar et al. do provide lower bounds for both deterministic and randomized algorithms. The most appropriate to compare to the rest of this paper is the Monte Carlo algorithm that succeeds with error less than $\frac{1}{k}$ which requires $O(k \log^2 n - log(1-\delta))$ bits in order to only count 0's and 1's on a sliding window.

From this point, they further provide the lower bound for the Sum algorithm, which is a modification of Basic Counting. This bound is $O(\log n(\log n + \log m) = O(\log^2(m)$. The authors note that this bound is a special case of the L_p norm, which has asymptotic space equivalent to F_2, in which the vector is one dimensional.[1] However, this seems very strange, as it would imply that the problem is the same asymptotic difficulty as solving L_2 on the general streaming model. Given the increased error by implicit deletion, it would be very strange if the two functions had identical lower bounds, when error is considered as a parameter (especially when it can be as large as sublinear on n). We believe then, that it remains to be proven what the exact lower bound required to provide a $(1+\epsilon)$ approximation for F_2 for streams with non constant epsilon.

9 Ideas Considered

The data structures provided in the papers we evaluated are complex and carefully analyzed, however they are also designed to cover a wide range of similar functions. Our goal when attempting to solve the problem ourselves was to focus closely on the F_2 function and attempt to leverage the nature of the probem to yield an improved algorithm.

One of the ideas considered was to attempt to approximate the function not by merging buckets to bound error, but by simply running individual instances of the F_2 algorithm of Alon, Matias, and Szegedy (AMS). However, this is problematic in the worst case.

Lemma 9.1. *In the worst case, the growth rate of the F_2 would require a polynomial amount of AMS sketches to approximate*

Proof. For maximum function growth, assume every element is the same, and let $|D| = n$, $F_{2,D} = n^2$. Adding one more element, $F_{2,D+1} = n + 1^2$. Subtracting the two, we get $2n + 1$ growth on F_2 per element added. Thus, if we maintain a polylogarithmic amount of sketches, we must create a sketch every $\frac{N}{\log^{O(1)}}$ elements. Thus, we would have a growth on the order of $\frac{N^2}{\log^{2O(1)}}$ in between sketches. The minimum growth is if all elements are new and different between the sketches, which would yield a growth of $\frac{N}{\log^{O(1)}}$. Thus, this is infeasible for small growth for constant or small epsilon as the difference in the size of F_2 between minimum and maximum is of order N^2. □

However, given sub-linear space, a constant approximation could be obtained, however this is much worse than current bounds!

[1] They also claim that the algorithm they produce provides an asymptotic matching upper bound. They do not, however, provide a $(1+\epsilon)$ approximation.

10 Conclusions

While we were unable to provide an improved algorithm, or improved calculations for the lower bound, we believe we highlight the importance of carefully analyzing published results to determine the information contained within. In this case, we saw that even though the authors found a matching upper and lower bound in terms of the length of the window, the algorithm was later improved, and possibly could be improved further. We still believe that with another bucket based data structure, this one more specifically tailored to just frequency moments, this upper bound could be further improved in terms of ϵ.

Additionally, we consider the difficulty of translating analysis from one streaming model to another, and ask if with improved analysis if, given the matching lower bounds for F_2 in both models, if algorithms with equal asymptotic space requirements are possible, thus showing the models are only seperated by a constant factor.

References

[1] Noga Alon, Yossi Matias, Mario Szegedy February 22, 2002 *The space complexity of approximating the frequency moments.*

[2] Mayur Datar, Aristides Gionis, Piotr Indyk, Rajeev Motwani 2002 *Maintaining Stream Statistics Over Sliding Windows* [p 1794-1813] *Siam J. Comput.*

[3] Names here year here *Title here*