Maintaining Variance and k–Medians over Data Stream Windows

Brian Babcock [*] Mayur Datar [†] Rajeev Motwani [‡] Liadan O'Callaghan [§]

Department of Computer Science
Stanford University
Stanford, CA 94305
{babcock, datar, rajeev, loc}@cs.stanford.edu

ABSTRACT

The sliding window model is useful for discounting stale data in data stream applications. In this model, data elements arrive continually and only the most recent N elements are used when answering queries. We present a novel technique for solving two important and related problems in the sliding window model — maintaining variance and maintaining a k–median clustering. Our solution to the problem of maintaining variance provides a continually updated estimate of the variance of the last N values in a data stream with relative error of at most ϵ using $O(\frac{1}{\epsilon^2}\log N)$ memory. We present a constant-factor approximation algorithm which maintains an approximate k–median solution for the last N data points using $O(\frac{k}{\tau^4}N^{2\tau}\log^2 N)$ memory, where $\tau < 1/2$ is a parameter which trades off the space bound with the approximation factor of $O(2^{O(1/\tau)})$.

1. INTRODUCTION

The *data stream* model of computation [14] is useful for modeling massive data sets (much larger than available main memory) that need to be processed in a single pass. Motivating applications include networking (traffic engineering, network monitoring, intrusion detection), telecommunications (fraud detection, data mining), financial services (arbitrage, financial monitoring), e-commerce (clickstream analysis, personalization), and sensor networks. This model, in which the data is treated as a possibly infinite sequence of records, captures many of the characteristics of such massive data sets, as distinct from standard data sets that can be loaded into a traditional RDBMS and analyzed offline.

Data streams are of unbounded length, so it is impossible to guarantee storage of the entire data set; however, for many applications, it is important to be able to execute queries that reference more data than can fit in main memory. Making such queries possible is a major challenge of data stream processing. Given memory constraints, it becomes necessary to devise techniques for maintaining summaries or synopses of the history of the stream. Most research until now has focused on techniques for building summaries of all the data seen so far. However, in most applications very old data points are considered less useful and relevant than more recent data. Thus it is necessary that old or "stale" data does not overly influence the statistics or models that are built from the analysis. There are two common approaches to deal with the problem of staleness of historical data. One method is *aging* — data elements are associated with weights that decrease over time. In most algorithms that use aging, the weights decrease according to the computationally-simple exponential-decay model (e.g. Gilbert et al. [11]), although linear-decay models are also used. The *sliding window* model [2, 7] is the other commonly-used mechanism for discounting stale data. Here, only the last N elements to arrive in the stream are considered relevant for answering queries, where N is the *window size*.

Datar, Gionis, Indyk, and Motwani [7] considered the problem of maintaining simple statistics over sliding windows, which is an important tool for processing data stream queries. They presented a general framework based on a novel data structure called an *exponential histogram (EH)* to estimate a class of aggregate functions over sliding windows. Their result applies to any function f satisfying the following properties for all multisets X, Y:

Property 1. $f(X) \geq 0$.

Property 2. $f(X) \leq poly(|X|)$.

Property 3. $f(X \cup Y) \geq f(X) + f(Y)$.

Property 4. $f(X \cup Y) \leq C_f(f(X)+f(Y))$, where $C_f \geq 1$ is a constant.

Furthermore, for their technique to be efficient, it must be possible to compute (or at least estimate) $f(X)$ from a small

[*]Supported in part by a Rambus Corporation Stanford Graduate Fellowship and NSF Grant IIS-0118173.

[†]Supported in part by Siebel Scholarship and NSF Grant IIS-0118173.

[‡]Supported in part by NSF Grant IIS-0118173, an Okawa Foundation Research Grant, an SNRC grant, and grants from Microsoft and Veritas.

[§]Supported in part by an NSF Graduate Fellowship, an ARCS Fellowship, and NSF Grant IIS-0118173.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
PODS 2003, June 9-12, 2003, San Diego, CA.
Copyright 2003 ACM 1-58113-670-6/03/06 ...$5.00.

sketch of X, where a sketch is a synopsis structure of small size which is *additive* — given the sketches for two multi-sets X and Y, a sketch for $X \cup Y$ can be quickly computed. They observe that *sum* and l_1/l_2 vector norms satisfy these properties. However, many interesting statistics do not satisfy these properties and cannot be estimated using their techniques.

In this paper, we expand the applicability of the EH technique, extending it to work for statistics that do not obey the above properties. We consider two important statistics — variance and k–median. Variance is a fundamental statistical measure, widely used for data analysis. The k–median clustering problem has been studied extensively across several disciplines and is an effective means of summarizing data or building a model for visualization and analysis. For example, clustering data in a customer-information database may help advertisers discover market segments, and clustering telephone-call records may expose fraudulent telephone use. In the systems community, k–median [4, 21] and clustering in general [13, 20, 22] have long been areas of active research. Previous stream clustering algorithms aim to maintain clusters that are valid for all points since the beginning of the stream; that is, old points do not expire. As such, these algorithms do not address the important issue of *concept drift*, a major problem in online learning caused when a model based on old data fails to correctly reflect the current state of the world. Domingos, Hulten and Spencer [9] have used sliding windows to deal with this problem in the context of learning decision trees over data streams. One contribution of our work is to begin to address the problem of concept drift by providing the first data stream clustering algorithm to work over sliding windows for k–median clustering.

Note that the variance is nothing more than the k–median cost for a special case of (continuous) k–median, where $k = 1$, data points are from a one-dimensional space, and the *sum of squared distances* metric is used. This similarity between the two translates into a common technique for computing both; it may be possible to use this technique in computing other similar statistics or synopses over sliding windows. The technique that we use to solve both problems is to summarize intervals of the data stream using composable synopses. In order to make efficient use of memory, synopses for adjacent intervals are combined when doing so will not increase the relative error significantly. The synopsis for the interval that straddles the sliding window boundary is inaccurate because some of the points it summarizes no longer fall within the sliding window and have expired. But we are able to contain this inaccuracy by ensuring that the size of this interval, in terms of the objective function that we are trying to estimate (variance or k–median clustering cost), is not large as compared to all other intervals. Moreover—and this is the main technical innovation that distinguishes our algorithms from the ones described by Datar, Gionis, Indyk, and Motwani [7]—we can estimate the contribution of this interval by treating its expired points as though they were "typical" points from the interval.

Our main results are as follows. We show how to estimate variance over sliding windows with relative error at most ϵ $(0 < \epsilon < 1)$ using $O(\frac{1}{\epsilon^2} \log N)$ memory. Further, we present an algorithm for k–median clustering over sliding windows using $O(\frac{k}{\tau^4} N^{2\tau} \log^2 N)$ memory. This is a constant-factor bicriteria approximation algorithm for the k–median problem — it uses $2k$ centers, and the objective function value is within a constant factor $(2^{O(1/\tau)})$ of optimal, where $\tau < 1/2$ is a parameter which captures the trade-off between the space bound and the approximation ratio. We build upon this solution to devise an algorithm using *exactly* k centers and providing the same approximation guarantee of $(2^{O(1/\tau)})$ and having the same asymptotic memory requirement $(O(\frac{k}{\tau^4} N^{2\tau} \log^2 N))$.

For k–median clustering we introduce a sliding-window algorithm that incorporates techniques from the one-pass data-stream clustering algorithm of Guha, Mishra, Motwani, and O'Callaghan [12]. Their algorithm uses $O(n^\tau)$ memory and provides a constant factor $(2^{O(1/\tau)})$ approximation to the k–median problem. Our sliding window algorithm also makes use of EH to estimate the value of the k–median objective function, though direct application of the techniques from Datar, Gionis, Indyk, and Motwani [7] is impossible (as in variance) because the k–median objective function does not satisfy Property 4 from above. If there are two sets of data points, each tightly clustered about its mean, then the variance of each set will be small; however, if the means of the two sets are far apart, then the variance of their union can be arbitrarily large. The same is true for clustering: although each of two sets of points may cluster nicely into k clusters, if the two sets of cluster centers are far apart, the union of the two sets is difficult to cluster. We demonstrate a new technique that gets around this problem and enables us to maintain EHs with small space and low relative error.

1.1 Related Work

Algorithms for streaming data have been an area of much recent research interest. A detailed survey of the algorithmic and database research in data streams is available [2]. Domingos, Hulten and Spencer [8, 9] study the problem of maintaining decision trees over sliding windows on data streams. Our results on k–median extends earlier work of Guha, Mishra, Motwani, and O'Callaghan [12], on one-pass clustering of data streams, to the sliding window model. There is a large body of previous work on k–median clustering [1, 5, 15, 16, 17, 18, 19]. Datar, Gionis, Indyk and Motwani [7] have considered the problem of maintaining simple statistics over sliding windows. Our work can be considered an extension of that work; we estimate functions over sliding windows that cannot be estimated using their techniques. Babcock, Datar, and Motwani [3] study sampling in the sliding window model.

In a recent paper, Gibbons and Tirthapura [10] improved the results from Datar et al. [7] for computing counts and sums over sliding windows. They present a new data structure called *waves* that has a worst-case update time of $O(1)$ compared to $O(\log N)$ for the EH data structure, and they also present three extensions of the sliding window model for the distributed scenario. The efficiency of the *waves* data structure crucially hinges on the additivity of the function f that we are estimating over sliding windows, i.e., in terms of the notation introduced in Section 1, f should satisfy $f(X \cup Y) = f(X) + f(Y)$. For this reason, the *waves* data structure does not appear to be applicable to the non-additive functions we consider in this paper.

1.2 Model and Summary of Results

In the *sliding window model*, data elements arrive in a stream and only the last N elements to have arrived are

considered relevant at any moment. These most recent N elements are called *active* data elements; the rest are called *expired* and they no longer contribute to query answers or statistics on the data set. Once a data element has been processed, it cannot be retrieved for further computation at a later time, unless it is explicitly stored in memory. The amount of memory available is assumed to be limited, in particular, sublinear in the size of the sliding window. Therefore algorithms that require storing the entire set of active elements are not acceptable within this model.

We employ the notion of a *timestamp*, which corresponds to the position of an active data element in the current window. We timestamp the active data elements from most recent to oldest with the most recently arrived data element having a timestamp of 1. Let x_i denote the data element with timestamp i. Clearly, the timestamps change with every new arrival, and we do not wish to make explicit updates. A simple solution is to record the arrival times in a wraparound counter of $\log N$ bits; then the timestamp can be extracted by comparison with the counter value of the current arrival.

We will maintain a histogram for the active data elements in the data stream. Our notion of histograms is far more general than the traditional one used in the database literature. In particular, every bucket in our histograms stores some summary/synopsis structure for a contiguous set of data elements, i.e., the histogram is partitioned into buckets based on the arrival time of the data elements. Along with this synopsis, for every bucket, we keep the timestamp of the most recent data element in that bucket (the *bucket timestamp*). When the timestamp of a bucket reaches $N+1$, all data elements in the bucket have expired, so we can drop that bucket and reclaim its memory. The buckets are numbered $B_1, B_2, \ldots, B_m$, starting from most recent (B_1) to oldest (B_m); further, $t_1, t_2, \ldots, t_m$ denote the bucket timestamps. All buckets, save the oldest, contain only active elements, while the oldest bucket may contain some expired elements besides at least one active element.

We now formally state the problems considered and describe our results in detail.

PROBLEM 1 (**Variance**). *Given a stream of numbers, maintain at every instant the variance of the last N values,*

$$VAR = \sum_{i=1}^{N} (x_i - \mu)^2,$$

where $\mu = \frac{1}{N}\sum_{i=1}^{N} x_i$ denotes the mean of the last N values.

Unless we decide to buffer the entire sliding window in memory, we cannot hope to compute the variance exactly at every instant. In Section 2 we present a small-space algorithm to solve this problem approximately. Our algorithm uses $O(\frac{1}{\epsilon^2}\log N)$ memory and provides an estimate at every instant that has relative error at most ϵ. The time required per new element is amortized $O(1)$.

We then extend our work to the k–median clustering problem. Given a multiset X of objects in a metric space M with distance function ℓ, the *k–median problem* is to pick k points $c_1, \ldots, c_k \in M$ so as to minimize $\sum_{x\in X} \ell(x, C(x))$, where $C(x)$ is the closest of $c_1, \ldots, c_k$ to x.[1] If $c_i = C(x)$ then x is said to be *assigned* to c_i, and $\ell(x, c_i)$ is called the *assignment distance* of x. The objective function is the sum of assignment distances.

PROBLEM 2 (**SWKM**). *Given a stream of points from a metric space M with distance function ℓ, window size N, and parameter k, maintain at every instant t a median set $c_1, c_2, \ldots, c_k \in M$ minimizing $\sum_{x\in X_t} \ell(x, C(x))$, where X_t is the multiset of the N most recent points at time t.*

In Section 3 we show how to maintain an approximate k–median solution over sliding windows using $\tilde{O}(\frac{k}{\tau^4}N^{2\tau})$ memory, for any $\tau < 1/2$, using amortized $\tilde{O}(k)$ insertion time per data point. We assume that the space required to store any point from the metric space and also the time required to compute the distance between any pair of points is $O(1)$. Strictly speaking, if the points belong to a d-dimensional space, both these quantities will be $O(d)$ and a multiplicative factor d will apply to both the memory and time requirement. But we ignore this for the rest of our discussion. The algorithm produces k medians such that the sum of assignment distances of the points in X_t is within a constant multiplicative factor $2^{O(1/\tau)}$ of the optimal.

2. MAINTAINING VARIANCE OVER SLIDING WINDOWS

Datar, Gionis, Indyk, and Motwani [7] presented a space lower bound of $\Omega(\frac{1}{\epsilon}\log N(\log N + \log R))$ bits for approximately (with error at most ϵ) maintaining the sum, where N is the sliding window size and each data value is at most R. Assuming $R = poly(N)$, this translates to a lower bound of $\Omega(\frac{1}{\epsilon}\log N)$ words for maintaining the sum, and hence the variance, of the last N elements. There is a gap of $\frac{1}{\epsilon}$ between this lower bound and the upper bound obtained here.

We now describe our algorithm for solving Problem 1, i.e., to compute an estimate of the variance with relative error at most ϵ. As mentioned before, elements in the data stream are partitioned into buckets by the algorithm. For each bucket B_i, besides maintaining the timestamp t_i of the most recent data element in that bucket, our algorithm maintains the following summary information: the number of elements in the bucket (n_i), the mean of the elements in the bucket (μ_i), and the variance of the elements in the bucket (V_i). The actual data elements that are assigned to a bucket are not stored.

In addition to the buckets maintained by the algorithm, we define another set of *suffix buckets*, denoted $B_{1^*}, \ldots, B_{j^*}$, that represent suffixes of the data stream. Bucket B_{i^*} represents all elements in the data stream that arrived after the elements of bucket B_i, that is, $B_{i^*} = \bigcup_{l=1}^{i-1} B_l$. Except for the bucket B_{m^*}, which represents all points arriving after the oldest non-expired bucket, these suffix buckets are not maintained by the algorithm, though their statistics are calculated temporarily during the course of the algorithm. The pseudocode for maintaining the histogram as new elements arrive is presented in Algorithm 1.

Combination Rule: While maintaining the histogram, our algorithm sometimes needs to combine two adjacent buckets. Consider two buckets B_i and B_j that get combined to form a new bucket $B_{i,j}$. The statistics (count, mean, and variance)

[1] This formulation of k–median is called *continuous k–median*. In *discrete k–median*, the medians must be chosen from the set X. For us, "k–median" will refer to the continuous version.

Algorithm 1 (Insert): x_t denotes the most recent element.

1. If $x_t = \mu_1$, then extend bucket B_1 to include x_t, by incrementing n_1 by 1. Otherwise, create a new bucket for x_t. The new bucket becomes B_1 with $V_1 = 0$, $\mu_1 = x_t$, $n_1 = 1$. An old bucket B_i becomes B_{i+1}.
2. If the oldest bucket B_m has timestamp greater than N, delete the bucket. Bucket B_{m-1} becomes the new oldest bucket. Maintain the statistics of B_{m-1^*} (instead of B_{m^*}), which can be computed using the previously maintained statistics for B_{m^*} and statistics for B_{m-1}.
3. Let $k = \frac{9}{\epsilon^2}$ and $V_{i,i-1}$ denote the variance of the bucket obtained by combining buckets B_i and B_{i-1}. While there exists an index $i > 2$ such that $kV_{i,i-1} \leq V_{i-1^*}$, find the smallest such i and combine buckets B_i and B_{i-1} using the combination rule described below. Note that the statistics for B_{i^*} can be computed incrementally from the statistics for B_{i-1} and B_{i-1^*}.
4. Output estimated variance at time t according to the estimation procedure below.

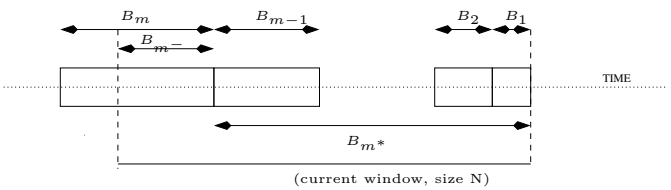

Figure 1: An illustration of the histogram.

for $B_{i,j}$ are computed from the statistics of B_i and B_j as follows: $n_{i,j} = n_i + n_j$; $\mu_{i,j} = \frac{\mu_i n_i + \mu_j n_j}{n_{i,j}}$; and, $V_{i,j} = V_i + V_j + \frac{n_i n_j}{n_{i,j}}(\mu_i - \mu_j)^2$.

Note that the combination rule can also be used to "delete" a set of points (A) from a larger set ($B \supseteq A$), i.e., calculate the statistics corresponding to the difference ($B - A$), based on the statistics for the two sets A, B. The following lemma shows the correctness of the statistics computed by the combination rule.

LEMMA 1. *The bucket combination procedure correctly computes $n_{i,j}$, $\mu_{i,j}$, and $V_{i,j}$ for the new bucket.*

PROOF. First, note that $n_{i,j}$ and $\mu_{i,j}$ are correctly computed by the definitions of count and average. Define $\delta_i = \mu_i - \mu_{i,j}$ and $\delta_j = \mu_j - \mu_{i,j}$.

$$
\begin{aligned}
V_{i,j} &= \sum_{x_l \in B_{i,j}} (x_l - \mu_{i,j})^2 \\
&= \sum_{x_l \in B_i} (x_l - \mu_i + \delta_i)^2 + \sum_{x_l \in B_j} (x_l - \mu_j + \delta_j)^2) \\
&= \sum_{x_l \in B_i} (x_l - \mu_i)^2 + 2\delta_i(x_l - \mu_i) + {\delta_i}^2 + \\
&\quad \sum_{x_l \in B_j} (x_l - \mu_j)^2 + 2\delta_j(x_l - \mu_j) + {\delta_j}^2 \\
&= V_i + V_j + {\delta_i}^2 n_i + {\delta_j}^2 n_j + \\
&\quad 2\delta_i \left(\sum_{x_l \in B_i} x_i - \sum_{x_l \in B_i} \mu_i \right) + \\
&\quad 2\delta_j \left(\sum_{x_l \in B_j} x_j - \sum_{x_l \in B_j} \mu_j \right) \\
&= V_i + V_j + (\delta_i)^2 n_i + (\delta_j)^2 n_j \\
&= V_i + V_j + n_i \left(\frac{n_j(\mu_j - \mu_i)}{n_i + n_j} \right)^2 + \\
&\quad n_j \left(\frac{n_i(\mu_i - \mu_j)}{n_i + n_j} \right)^2 \\
&= V_i + V_j + \frac{n_i n_j}{n_{i,j}} (\mu_i - \mu_j)^2
\end{aligned}
$$

□

Estimation: Let $B_1, \ldots, B_m$ be the set of histogram buckets at time t. We describe a procedure to estimate the variance over the current active window. Let B_m be the oldest active bucket. It contains some active elements, including the one with timestamp N, but may also contain expired data. We maintain statistics corresponding to B_{m^*}, the suffix bucket containing all elements that arrived after bucket B_m. To this end, we use the combination rule for every new data element that arrives. Whenever the oldest bucket gets deleted, we can find the new B_{m^*} by "deleting" the contribution of the new oldest active bucket (B'_m) from the previous B_{m^*}, using the combination rule.

Let $B_{\tilde{m}}$ refer to the non-expired portion of the bucket B_m, i.e., the set of elements in B_m that are still active. (See Figure 1 for an illustration.) Since we do not know the statistics $n_{\tilde{m}}$, $\mu_{\tilde{m}}$, and $V_{\tilde{m}}$ corresponding to bucket $B_{\tilde{m}}$, we estimate them as follows: $n_{\tilde{m}}^{EST} = N + 1 - t_m$; $\mu_{\tilde{m}}^{EST} = \mu_m$; and $V_{\tilde{m}}^{EST} = \frac{V_m}{2}$. Note that $n_{\tilde{m}}$ is exact, i.e., $n_{\tilde{m}}^{EST} = n_{\tilde{m}}$.

The statistics for $B_{\tilde{m}}$ and B_{m^*} are sufficient to accurately compute the variance at the time instant t. In fact the variance is nothing but the variance corresponding to the bucket $B_{\tilde{m},m^*}$ obtained by combining $B_{\tilde{m}}$ and B_{m^*}. Therefore, by the combination rule, the actual variance ($\widehat{\mathrm{VAR}}(t)$) for the current active window is given by:

$$\widehat{\mathrm{VAR}}(t) = \left(V_{\tilde{m}} + V_{m^*} + \frac{n_{\tilde{m}} n_{m^*}}{n_{\tilde{m}} + n_{m^*}} (\mu_{\tilde{m}} - \mu_{m^*})^2 \right).$$

At every time instant t we estimate the variance by computing the variance of $B_{\tilde{m},m^*}$ using the estimates above for $B_{\tilde{m}}$. This estimate can be found in $O(1)$ time provided we maintain the statistics for $B_{\tilde{m}}$ and B_{m^*}. The error in our estimate arises due to the error in the estimate of the statistics for $B_{\tilde{m}}$. As we shall prove below, this error is small

(within factor ϵ) as compared to the exact answer ($\widehat{\mathrm{VAR}}(t)$) provided we maintain the following invariant:

INVARIANT 1. *For every bucket B_i, $\frac{9}{\epsilon^2}V_i \leq V_{i^*}$.*

It is easy to see that our algorithm maintains the invariant above. Any bucket will have a non-zero variance only if it is formed by the combination of two buckets, and the condition for combining two buckets guarantees that the invariant holds for the combined bucket when it is formed. Once the invariant holds for a bucket B_i it continues to hold in the future since the variance of the suffix bucket B_{i^*} is non decreasing with time. Additionally, our algorithm maintains the following invariant which ensures that the total number of buckets is small:

INVARIANT 2. *For each $i > 1$, for every bucket B_i,*

$$\frac{9}{\epsilon^2}V_{i,i-1} > V_{i-1^*}.$$

LEMMA 2. *The number of buckets maintained at any point in time by an algorithm that preserves Invariant 2 is*

$$O(\frac{1}{\epsilon^2}\log NR^2),$$

where R is an upper bound on the absolute value of the data elements.

PROOF SKETCH. It follows from the combination rule that the variance for the union of two buckets is no less than the sum of the individual variances. Therefore, it is easy to see that, for any algorithm that preserves Invariant 2 the variance of the suffix bucket B_{i^*} doubles after every $O(\frac{1}{\epsilon^2})$ buckets. Thus, the total number of buckets maintained by such an algorithm is no more than $O(\frac{1}{\epsilon^2}\log V)$, where V is the variance of the last N points. If R is an upper bound on the absolute value of the data elements, V is no more than NR^2 and the claim follows. □

Note that we require $\Omega(\log R)$ bits of memory to represent each data element. We will need the following technical lemma to analyse the performance of our algorithm.

LEMMA 3. *For any choice of a and any set of data elements B with mean μ, $\sum_{x\in B}(x-a)^2 = \sum_{x\in B}(x-\mu)^2 + |B|(a-\mu)^2$*

PROOF. The proof follows from the following calculations.

$$\begin{aligned}
\sum_{x\in B}(x-\mu)^2 &+ |B|(a-\mu)^2 \\
&= \sum_{x\in B}(x-\mu)^2 + (a-\mu)^2 \\
&= \sum_{x\in B} x^2 + 2\mu^2 + a^2 - 2x\mu - 2a\mu \\
&= \sum_{x\in B} x^2 + 2\mu(\mu - x) - 2a\mu + a^2 \\
&= \sum_{x\in B} x^2 - 2ax + a^2 \\
&= \sum_{x\in B}(x-a)^2.
\end{aligned}$$

□

The following theorem summarizes the algorithm's performance.

THEOREM 1. *Let $VAR(t)$ be the variance estimate provided by the algorithm maintaining Invariants 1 and 2, and let $\widehat{VAR}(t)$ be the actual variance. Then $(1-\epsilon)\widehat{VAR}(t) \leq VAR(t) \leq (1+\epsilon)\widehat{VAR}(t)$. Further, this algorithm requires at most $O(\frac{1}{\epsilon}\log NR^2)$ memory.*

PROOF. The memory usage is demonstrated by Lemma 2. Define $\delta = \mu_m - \mu_{\tilde{m}} = \mu_{\tilde{m}}^{EST} - \mu_{\tilde{m}}$. By the combination rule and our estimates for $B_{\tilde{m}}$,

$$\begin{aligned}
\mathrm{VAR}(t) &- \widehat{\mathrm{VAR}}(t) \\
&= (V_{\tilde{m}}^{EST} + V_{m^*} + \frac{n_{\tilde{m}}n_{m^*}}{n_{\tilde{m}}+n_{m^*}}(\mu_{\tilde{m}}^{EST} - \mu_{m^*})^2) - \\
&\quad (V_{\tilde{m}} + V_{m^*} + \frac{n_{\tilde{m}}n_{m^*}}{n_{\tilde{m}}+n_{m^*}}(\mu_{\tilde{m}} - \mu_{m^*})^2) \\
&= (V_m/2 - V_{\tilde{m}}) + \\
&\quad \frac{n_{\tilde{m}}n_{m^*}}{n_{\tilde{m}}+n_{m^*}}(2\delta(\mu_{\tilde{m}} - \mu_{m^*}) + \delta^2) \\
&= (V_m/2 - V_{\tilde{m}}) + \frac{n_{\tilde{m}}n_{m^*}}{n_{\tilde{m}}+n_{m^*}}\delta^2 + \\
&\quad \frac{n_{\tilde{m}}n_{m^*}}{n_{\tilde{m}}+n_{m^*}}(2\delta(\mu_{\tilde{m}} - \mu_{m^*})).
\end{aligned}$$

We will show that each of the three additive terms in the error is small. Since $B_{\tilde{m}}$ is a subinterval of B_m we know that $V_{\tilde{m}} \leq V_m$. As variance is always non-negative, it follows that $|\frac{V_m}{2} - V_{\tilde{m}}| \leq \frac{V_m}{2}$. By Lemma 3 we know that $n_{\tilde{m}}(\mu_{\tilde{m}} - \mu_{\tilde{m}}^{EST})^2 = n_{\tilde{m}}(\mu_{\tilde{m}} - \mu_m)^2 \leq \sum_{x\in B_{\tilde{m}}}(x-\mu_m)^2 \leq V_m$, which implies that $|\frac{n_{\tilde{m}}n_{m^*}}{n_{\tilde{m}}+n_{m^*}}\delta^2| \leq V_m$. Define $c_2 = 3$, a constant derived from the analysis. For the third error term, consider two cases:

Case 1 ($|\mu_{\tilde{m}} - \mu_{m^*}| \leq |\frac{c_2\delta}{\epsilon}|$): Then

$$\begin{aligned}
|\frac{n_{\tilde{m}}n_{m^*}}{n_{\tilde{m}}+n_{m^*}}2\delta(\mu_{\tilde{m}} - \mu_{m^*})| &\leq \frac{n_{\tilde{m}}n_{m^*}2c_2\delta^2}{(n_{\tilde{m}}+n_{m^*})\epsilon} \\
&\leq \frac{2c_2V_m}{\epsilon}.
\end{aligned}$$

Case 2 ($|\mu_{\tilde{m}} - \mu_{m^*}| > |\frac{c_2\delta}{\epsilon}|$): The actual variance $\widehat{\mathrm{VAR}}(t)$ is at least $\frac{n_{\tilde{m}}n_{m^*}}{n_{\tilde{m}}+n_{m^*}}(\mu_{\tilde{m}} - \mu_{m^*})^2$. The ratio between the error term and the overall variance is at most

$$\frac{|2\delta(\mu_{\tilde{m}} - \mu_{m^*})|}{(\mu_{\tilde{m}} - \mu_{m^*})^2} \leq |\frac{2\delta}{\mu_{\tilde{m}} - \mu_{m^*}}| \leq \frac{2\epsilon}{c_2}.$$

By Invariant 1, we know that $V_m \leq \frac{\epsilon^2}{9}V_{m^*}$. The first two error terms contribute a combined additive error of at most $\frac{3}{2}V_m$, which is a multiplicative error of at most $\frac{1}{6}\epsilon^2$ since $V_{m^*} \leq \widehat{\mathrm{VAR}}(t)$. The third error term contributes an additive error of at most $\frac{2c_2V_m}{\epsilon}$ (in case 1), which represents a multiplicative error of at most $\frac{2}{3}\epsilon$. In case 2 the multiplicative error from the third error term is at most $\frac{2}{3}\epsilon$. We assume that $\epsilon \leq 1$ since otherwise the trivial algorithm that always returns zero suffices. In both cases we have the total error strictly less than an ϵ fraction of the overall variance, proving the theorem. □

The algorithm presented earlier requires $O(\frac{1}{\epsilon^2}\log NR^2)$ time per new element. Most of the time is spent in Step 3,

where we make the sweep to combine buckets. This time is proportional to the size of the histogram ($O(\frac{1}{\epsilon^2}\log NR^2)$). A simple trick is to skip Step 3 until we have seen $\Theta(\frac{1}{\epsilon^2}\log NR^2)$ data points. This ensures that the running time of the algorithm is amortized $O(1)$. While we may violate Invariant 2 temporarily, we restore it every $\Theta(\frac{1}{\epsilon^2}\log NR^2)$ data points when we execute Step 3. This ensures that the number of buckets is $O(\frac{1}{\epsilon^2}\log NR^2)$.

3. CLUSTERING ON SLIDING WINDOWS

3.1 Bicriteria Approximate Clustering

We now present a solution to the SWKM problem (Problem 2). As mentioned earlier, we focus on continuous k–median; however, the techniques also apply to discrete k–median, although the approximation ratios will be different. Our solution incorporates the techniques of Guha, Mishra, Motwani, and O'Callaghan [12], but ensures a high-quality sliding-window clustering. Roughly speaking, the algorithm of Guha et al. is a divide-and-conquer algorithm that builds clusters incrementally and hierarchically as the stream grows. The algorithm maintains medians (cluster centers) at various levels, the original input points being considered to be at level 0. On seeing n^τ level-0 medians (input points) it clusters them, using any bicriteria clustering algorithm, into $O(k)$ cluster centers which form the level-1 medians. When n^τ of these level-1 medians accumulate, they are clustered into level-2 medians and so on. In general, whenever there are n^τ medians at level-i they are clustered to form level-$(i+1)$ medians. Thus, at any moment their algorithm maintains at most n^τ medians at every level. Since we cluster groups of size n^τ, the hierarchical tree formed due to clustering has depth at most $1/\tau$ if there are n original input points. The authors prove that every level of the hierarchical tree increases the approximation factor by at most a constant multiple. As a result their algorithm has an approximation guarantee of $2^{O(1/\tau)}$ and a memory requirement of $\frac{1}{\tau}n^\tau$.

We now describe an algorithm for sliding-window k–median (SWKM) that uses the techniques from the above-described algorithm and ideas from the previous section. This section is organized as follows: first, we describe the EH data structure used by our algorithm and the method for associating a cost with each bucket; next, we describe the combination step for combining two buckets; then, we discuss the estimation procedure of the final clustering of the current active window after each data point arrives, or whenever a clustering is desired; and finally, we present the overall algorithm that maintains the data structure as new points arrive.

Data Structure: As in the previous case, the data structure maintained is an EH whose buckets are numbered $B_1, B_2, \ldots, B_m$ from most recent to oldest, and buckets containing only expired points are discarded. As with variance, each bucket stores a summary structure for a set of contiguous points as well as the timestamp of the most recent point in the bucket. In the case of variance, the summary structure contained the triplet (n_i, μ_i, V_i); for clustering, each bucket consists of a collection of data points or intermediate medians. For consistency, we will refer to the original points as level-0 medians.

Each median is represented by the triple $(p(x), w(x), c(x))$. The value $p(x)$ is the identifier of x; in Euclidean space, for example, this could be the coordinates of x, and in a general metric this could simply be the index of x in the point set. The value $w(x)$ is the weight of a median x, i.e., the number of points that x represents. Similar to the algorithm of Guha et al. [12], if x is of level 0, $w(x) = 1$, and if x is of level i, then $w(x)$ is the sum of the weights of the level-$(i-1)$ points that were assigned to x when the level-i clustering was performed. Finally, $c(x)$ is the estimated cost of x, i.e., an estimate of the sum of the costs $\ell(x, y)$ of assigning to x each of the leaves y of the subtree rooted at x. If x is of level 0, $c(x) = 0$; if x is of level 1, $c(x)$ is the sum of assignment distances of the members of x; and, if x is of level $i > 1$, $c(x)$ is the sum over all members y of x, of $c(y) + w(y) \cdot \ell(x, y)$. Thus, $c(x)$ is an overestimate of the "true" cost of x.

As in the algorithm of Guha et al. [12], we maintain medians at intermediate levels and whenever there are N^τ medians at the same level we cluster them into $O(k)$ medians at the next higher level. Thus, each bucket B_i can be split into $1/\tau$ (the maximum number of different levels) groups $R_i^0, \ldots, R_i^{1/\tau-1}$, where each R_i^j contains medians at level j. Each group contains at most N^τ medians. Along with a collection of medians, the algorithm also maintains for each bucket B_i a *bucket cost*, which is a conservative overestimate of the assignment cost of clustering all the points represented by the bucket.

Cost Function: The bucket cost function is an estimate of the assignment cost of clustering the points represented by the bucket. Consider a bucket B_i and let $Y_i = \bigcup_{j=0}^{1/\tau-1} R_i^j$ be the set of medians in the bucket. As explained earlier, each median x is associated with a cost $c(x)$ which is an estimate of the sum of the distances to x from all points assigned to x by the clustering algorithm. We cluster the points in Y_i to produce k medians $c_1, \ldots, c_k$. The cost function for bucket B_i is given by: $f(B_i) = \sum_{x \in Y_i} c(x) + w(x) \cdot \ell(x, C(x))$, where $C(x) \in \{c_1, \ldots, c_k\}$ is the median closest to x. Clearly, $f(B_i)$ is an overestimate of the assignment cost of clustering all the original points that are assigned to the medians in Y_i, using $c_1, \ldots, c_k$ as medians.

Combination: Let B_i and B_j be two (typically adjacent) buckets that need to be combined to form the merged bucket $B_{i,j}$, and let $R_i^0, \ldots, R_i^{1/\tau-1}$ and $R_j^0, \ldots, R_j^{1/\tau-1}$ be the groups of medians from the two buckets, where R_i^l (resp., R_j^l) represents the group of medians at level l in bucket B_i(resp., B_j). We set $R_{i,j}^0 = R_i^0 \bigcup R_j^0$. If $|R_{i,j}^0| > N^\tau$, then cluster the points from $R_{i,j}^0$ and set $R_{i,j}^0$ to be empty. Let C_0 denote the set of $O(k)$ medians obtained by clustering $R_{i,j}^0$, or the empty set if $R_{i,j}^0$ did not need to be clustered. We carry over these level-1 medians to the next group $R_{i,j}^1$. Set $R_{i,j}^1 = R_i^1 \bigcup R_j^1 \bigcup C_0$. As before, if there are more than N^τ medians, we cluster them to get a carry-over set C_1, and so on. In general, after at most $1/\tau$ unions, each possibly followed by clustering, we get the combined bucket $B_{i,j}$. Finally, the bucket cost is computed by clustering all medians in the bucket (at all levels).

Estimation: Let $B_1, B_2, \ldots, B_m$ denote the buckets at any time instant t. B_m is the oldest bucket and contains medians that represent some data points that have already expired. If a query is posed at this moment that asks for a clustering of the active elements we do the following:

- Consider all but the oldest of the buckets: $B_1, \ldots, B_{m-1}$. They each contain at most $\frac{1}{\tau}N^\tau$ medians. We will prove that the number of buckets is $O(\log N)$. Thus we have $O(\frac{1}{\tau}N^\tau \log N)$ medians. Cluster them to produce k medians.
- Similarly, cluster bucket B_m to produce k additional medians.
- Present the $2k$ medians as the answer.

If required, the procedure can also provide an estimate for the assignment cost using the same technique as that used for computing the cost function over buckets.

Algorithm: The algorithm for combining and maintaining buckets is very similar to that used for estimating variance. As before, we define suffix buckets B_{i^*} which represent the combination of all buckets that are later than a particular bucket (B_i). These are not maintained at all times but instead are computed when required, as in the case of variance. The pseudocode for our algorithm is presented below.

Algorithm 2 (Insert): x_t denotes the most recent element.

1. If there are fewer than k level-0 medians in B_1 add the point x_t as a level-0 median in bucket B_1. Otherwise, create a new bucket B_1 to contain x_t and renumber the existing buckets accordingly.
2. If bucket B_m has timestamp more than N, delete it.
3. Make a sweep over the buckets from most recent to least recent and while there exists an index $i > 2$ such that $f(B_{i,i-1}) \leq 2f(B_{i-1^*})$, find the smallest such i and combine buckets B_i and B_{i-1} using the combination procedure described above. The suffix bucket B_{i^*} is computed incrementally as we make the sweep.

Our algorithm maintains the following two invariants, which are useful in the proofs of Lemmas 4 and 5.

INVARIANT 3. *For every bucket B_i, $f(B_i) \leq 2f(B_{i^*})$.*

INVARIANT 4. *For every bucket B_i ($i > 1$), $f(B_{i,i-1}) > 2f(B_{i-1^*})$.*

LEMMA 4. *Because the algorithm maintains Invariant 3, it produces a solution with $2k$ medians whose cost is within a multiplicative factor of $2^{O(1/\tau)}$ of the cost of the optimal k–median solution.*

PROOF SKETCH. Let B_m be the oldest bucket in the histogram. Recall that B_m may contain some medians that represent points that have expired. Consider first the suffix bucket B_{m^*} representing all points arriving after the oldest non-expired bucket and compare our algorithm's performance on this set of points to the optimal solution on the same set of points.

We cluster medians at any level only when there are at least N^τ medians at that level, so that the depth of the hierarchical clustering tree is guaranteed not to exceed $\frac{1}{\tau}$. As discussed above, at each level in the tree, the divide-and-conquer approach introduces a constant multiplicative approximation factor. These approximation factors accumulate, so the overall clustering cost for our algorithm's clustering of B_{m^*} is $2^{O(1/\tau)}$ times the cost of the optimal k–median clustering of B_{m^*}.

Now consider the non-expired points from bucket B_m. Invariant 3 guarantees that $f(B_m) \leq 2f(B_{m^*})$. This means that $f(B_m)$, our algorithm's cost for clustering all of B_m, is at most twice $f(B_{m^*})$, which is within a $2^{O(1/\tau)}$ factor of the optimal clustering cost for B_{m^*}. Only the cost of clustering the non-expired portion of B_m counts against us, but this can only be less than the clustering cost when summing over B_m in its entirety.

Therefore the costs of our algorithm's solution for the points in B_{m^*} and also for the points in B_m are both within a $2^{O(1/\tau)}$ factor of the optimal clustering cost for B_{m^*}. B_{m^*} is a subset of the active points, so the cost of the optimal k–median clustering of all active points can only be greater than the cost of the optimal clustering of B_{m^*}. □

LEMMA 5. *Since the algorithm maintains Invariant 4, the number of buckets never exceeds $O(\frac{1}{\tau}\log N)$.*[2]

PROOF SKETCH. Invariant 4 guarantees that the number of buckets is at most $2\log R$ where R is our cost function over the entire sliding window. Lemma 4 proves that our cost function is at most $2^{O(1/\tau)}$ times the cost of the optimal k–median solution. Since the optimal k–median cost for N points is $poly(N)$ this gives us that $\log R = O(\frac{1}{\tau}\log N)$. □

The above k–median algorithm is efficient in terms of memory, but not in running time. After each element arrives, the algorithm checks all the buckets to see whether Invariant 4 is violated, in which case it combines two adjacent buckets to restore the invariant. In order to reduce the per-item processing time, we can use the same batch processing technique as in Section 2 to reduce the amortized time to $\tilde{O}(k)$. To this end, we draw a distinction between data structure maintenance and output production. We assume that the algorithm will not be called upon to actually produce an updated clustering as each new point is added. Instead, it may maintain sufficient statistics so as to be able, upon request, to quickly generate a valid clustering. Requests for the current clusters may come at arbitrary points in time, but we assume that they will not come too frequently. In short, we distinguish between update time and query time for our data structure.

We can modify Algorithm 2 so that it does not execute the bucket combination procedure until $\frac{k}{\tau^3}N^{2\tau}\log N$ points have accumulated in bucket B_1. No effort is made to maintain the invariants until the bucket "fills up," i.e. has more than $\frac{k}{\tau^3}N^{2\tau}\log N$ points, at which time the points in B_1 are clustered and replaced by k level-1 medians. However, the original data points are not yet discarded; they are retained until their bucket satisfies satisfy Invariant 3. (If this bucket is combined with another bucket, the resulting combined bucket will necessarily satisfy Invariant 3.) After clustering the points in B_1, we execute the remaining steps in the algorithm (discarding expired buckets and maintaining Invariant 4).

Consider what happens to the invariants as points accumulate in bucket B_1. Since the value of $f(B_{i^*})$ is increasing for all $i > 1$, it is possible that Invariant 4 may no longer be satisfied. However, the temporary failure of this invariant is not important; recall (Lemma 4) that Invariant 4 is not

[2]We assume that $\ell(x, y)$ is bounded by a polynomial in N.

essential for the correctness of the answer provided by the algorithm, but ensures that the number of buckets remains small. However, while the algorithm is filling up bucket B_1, the number of buckets does not increase, and as soon as the bucket is complete the invariant is restored by combining buckets as necessary. Thus, the number of buckets maintained by the algorithm is always $O(\frac{1}{\tau}\log N)$ and Lemma 5 remains valid even though Invariant 4 may temporarily get violated.

Invariant 3 will not cause trouble for any bucket B_i $(i > 1)$ as B_i fills, because the increase in $f(B_{i^*})$ for these buckets only strengthens the invariant. However, there may be buckets for which this invariant does not hold. The batch processing changes described above guarantee that for such buckets we retain the original data points. Recall from the proof of Lemma 4 that we used Invariant 3 in our analysis only to ensure that the bucket cost $f(B_m)$ of the oldest bucket was no more than twice the combined bucket cost $f(B_{m^*})$ and hence no more than $2^{O(1/\tau)}$ times the optimal clustering cost B_{m^*}. This was necessary because we could bound the performance of the original algorithm in terms of the clustering cost of bucket B_{m^*}, but not in terms of $f(B_m)$, because $f(B_m)$ potentially includes the cost of clustering expired points. However, as long as we maintain the original data points for every bucket B_i that violates Invariant 3, there is no problem. If the invariant holds for the oldest bucket B_m, then we use the estimation procedure described earlier; if the invariant fails for B_m, then given the original data points for the bucket, we can distinguish the expired elements from the active ones and cluster only the latter.

We cluster the level-0 medians using the randomized algorithm from Indyk [15], using the local search algorithm from Charikar and Guha [5] as a subroutine. This procedure requires linear space and takes time $\tilde{O}(nk)$ (where n is the number of points that are clustered) while providing a constant factor approximation with high probability. All higher level (level-1 and above) medians are clustered using the $O(n^2)$ local search algorithm of Charikar and Guha [5]. While this algorithm uses up to $2k$ centers, the number can be reduced to k for the final answer via the primal-dual algorithm of Jain and Vazirani [16].

Now consider the running time of the modified algorithm. Whenever a current clustering is desired, we cluster the medians in the oldest bucket B_m and the medians in the suffix buckets B_{m^*}. If bucket B_m violates Invariant 3 then we cluster only the active elements in it using the $\tilde{O}(nk)$ algorithm from Indyk [15] to produce k medians which are then clustered with the medians from the suffix bucket B_{m^*}. The total number of such medians is at most $O(\frac{1}{\tau^2}N^\tau \log N)$. The running time for clustering the active elements in B_m dominates, giving a total query time of $\tilde{O}(\frac{k}{\tau^4}N^{2\tau})$.

Putting everything together we have Theorem 2. The memory bound follows from Lemma 5. The space used to maintain the original data points for buckets violating Invariant 3 dominates the space used to maintain the EH itself. The approximation guarantee follows from Lemma 4, while the amortized maintenance time and query time follow from the discussion in the last two paragraphs.

THEOREM 2. *Our algorithm provides a $(2, 2^{O(1/\tau)})$ bicriteria approximation to the* SWKM *problem for any $\tau < 1/2$. It uses $O(\frac{k}{\tau^4}N^{2\tau}\log^2 N)$ memory and requires amortized $\tilde{O}(k)$ maintenance time per data element. The query time for the data structure that we maintain is $\tilde{O}(\frac{k}{\tau^4}N^{2\tau})$.*

We have described a bicriteria approximation algorithm that produces $2k$ centers as opposed to k. Next, we build upon this algorithm to get an algorithm that produces exactly k centers, while preserving the approximation guarantee of $2^{O(1/\tau)}$.

3.2 Producing Exactly k Clusters

In order to produce exactly k centers, we need to change the algorithm in two ways: (1) with each median, we maintain additional information that lets us estimate to within a constant factor the number of active data points that are assigned to it, and (2) we change the estimation procedure that produces a solution whenever a request is made for the current clusters. Rather than separately clustering the oldest bucket B_m and the suffix bucket B_{m^*}, we cluster the medians from all the buckets together. However, the weights of the medians from bucket B_m are adjusted so that they only reflect the contribution of active data points, discounting the contribution of the expired points.

The costs of B_m and B_{m^*} are both within $2^{O(1/\tau)}$ times the cost of the optimal k–median solution for B_{m^*} and hence within $2^{O(1/\tau)}$ times the cost of the optimal k–median solution for the current window. Moreover, the assignment cost for the active points in B_m is no more than the total cost of B_m. Using these facts and Theorem 2.3 from Guha et al. [12] it is easy to prove the following lemma for the modified estimation procedure above.

LEMMA 6. *Let $f(B_m) \le 2f(B_{m^*})$ be the cost of the oldest bucket. If we can assign weights to medians from B_m so that the weight of each median is within a constant factor c of the number of active points assigned to that median, then the optimal k–median solution for the instance of weighted medians from B_m and B_{m^*} is at most $f(B_m) + f(B_{m^*}) + cC^* = 2^{O(1/\tau)}C^*$, where C^* is the optimal k–median solution for the current window.*

PROOF SKETCH. The cost of assigning original points from B_{m^*} to their medians from B_{m^*} is no more than $f(B_{m^*}) \le 2^{O(1/\tau)}C^*$. The cost of assigning original points from B_m to their medians in B_m is no more than $f(B_m) \le 2f(B_{m^*}) \le 2^{O(1/\tau)}C^*$. Consider a k–median problem instance consisting of active original points from B_m and original points from B_{m^*}. This constitutes the current window and by definition the optimal clustering cost for this instance is C^*. Instead, if we assigned each original point a weight of c the cost of clustering this instance would be cC^*. The medians from B_m are weighted so that their weight is no more than c times the number of active points assigned to them. In other words, these can be thought of as medians for original points from B_m, where each original point has weight c. Applying Theorem 2.3 from Guha et al. [12], with $\ell = 2$, we get that the cost of the optimal k–median solution for the instance of weighted medians from B_m and B_{m^*} is at most $f(B_m) + f(B_{m^*}) + cC^* = 2^{O(1/\tau)}C^*$ □

In the case of estimating variance, we always knew exactly how points from the oldest bucket were active. Similarly, in the case of clustering, if we knew for each median in the oldest bucket how many active data points were assigned

to it, then we could cluster these medians along with medians from the suffix bucket B_{m^*} to get exactly k centers. In fact, the above lemma tells us that even if, instead of exact counts, we had only an estimate of the number of active points assigned to each median in B_m, we would get a constant factor approximation with exactly k centers.

We show how to estimate the number of active data points assigned to each median, to within a factor of 2, using at most $k^{1/\tau} \log N$ space per median. For any median c_j, consider the following imaginary stream $Z^{(j)}$ of $0-1$ values: For each actual data point x_i, $Z_i^{(j)} = 1$ if x_i was assigned to median c_j, and $Z_i^{(j)} = 0$ otherwise. Note that if the median c_j belongs to a particular bucket, $Z_i^{(j)}$ will be zero outside the interval covered by that bucket. Estimating the number of active data points assigned to c_j is the same as counting the number of ones from the last N elements of $Z^{(j)}$. If the stream $Z^{(j)}$ were presented explicitly, this problem could be solved using an EH data structure that uses $O(\frac{1}{\epsilon'} \log N)$ memory and provides an estimate at all times that has error at most ϵ', as shown by Datar et al. [7]. We set ϵ' equal to $1/2$ for our estimation. Unfortunately, the stream Z_j is not presented explicitly. Medians at level 2 and above are produced by the combination of lower-level medians. All points previously assigned to lower-level medians are now assigned to a higher-level median, but there is no way to reconstruct the exact arrival order (timestamps) of those points and simulate the stream $Z^{(j)}$ since the original points have been discarded. In order to estimate the number of active points for such higher-level medians we use the following two observations about EHs over $0-1$ data streams:

1. Consider two Exponential Histograms (EH1 and EH2) corresponding to two $0-1$ data streams for the same error parameter ϵ' and the same window size N. If all the 1's in EH1 have arrival times strictly greater than the arrival times for 1's in EH2 then we can combine the two EHs to get an EH for the union of the two streams in time $O(\log N)$. We say such EHs are *non-overlapping*. This "combination" can be easily achieved by placing the buckets of the two EHs one after the other and then making a sweep from right to left to combine the buckets. This process takes $O(\log N)$ time, and the combined EH uses $O(\log N)$ buckets or words of memory.

2. If the two data streams (respectively two EHs) are overlapping, i.e., they do not satisfy the property that all the 1's in one of them arrive before the 1's in the other, then we can maintain two separate EHs corresponding to them to answer count queries over the union of the two streams. The EHs corresponding to these overlapping streams are called *overlapping* EHs.

When a higher-level median is formed by the clustering of lower-level medians we assign all the EHs of the lower-level medians to the higher-level medians. We combine as many non-overlapping EHs as possible, but those that are overlapping are maintained separately. We claim that a median at level l needs to maintain at most k^{l-1} EHs, where each EH requires $O(\log N)$ memory. Since $\frac{1}{\tau}$ is the maximum level of any median, the amount of extra memory required per median is $O(k^{1/\tau} \log N)$.

We prove the claim by induction on the level of the medians. Original points (level-0 medians) do not maintain an EH; they simply maintain their timestamps. The $Z^{(j)}$'s corresponding to them have 1 in exactly one position and zero everywhere else. When level-0 medians are clustered to form k level-1 medians, for each level-1 median thus formed we can insert the level-0 medians assigned to it in the sorted (decreasing) order of timestamps so that we get a single EH for a level-1 median. Note that the k level-1 medians obtained in such a clustering may have overlapping EHs. A pair of level-i medians are called *conflicting* if they are created at the same time and have overlapping EHs. A level-i median can conflict with at most $k-1$ other level-i medians—the ones created during the same clustering of level-$(i-1)$ medians.

Consider what happens when we cluster level-1 medians to form a level-2 median. Of all the level-1 medians assigned to a single level-2 median, any given level-1 median can conflict with at most $k-1$ other level-1 medians. Consequently, we can organize the assigned medians into k groups such that within a group there are no conflicting medians. For each of these groups the corresponding EHs will be non-overlapping and can combined to form a single EH. Thus a level-2 median will have at most k EHs. By the inductive hypothesis, assume each level-l median has at most k^{l-1} overlapping EHs. By arguments same as above, when a level-$l+1$ median is formed, the level-l medians assigned to it can be organized into k groups such that within a group there are no conflicting medians. Moreover, since every level-l median within a group can contribute at most k^{l-1} overlapping EHs, we can combine all of them to form k^{l-1} overlapping EHs. Thus, across the k groups the level-$(l+1)$ median has a total of k^l EHs. This proves the claim.

The combination procedure for EHs described above is executed during the clustering of higher-level medians, i.e. medians from level 1 and above. It is done in the last phase when points are assigned to the cluster centers. The time taken to assign every point is now $O(k^{\frac{1}{\tau}} \log N)$ instead of $O(1)$. Since we use a quadratic running time algorithm to cluster the higher-level medians this does not affect the asymptotic running time for clustering provided $k^{\frac{1}{\tau}} \log N < N^\tau$. Moreover, the asymptotic space requirement of the algorithm also remains unchanged. This follows from considering the two different types of bucket: Buckets that violate Invariant 3 and retain the original points have $\frac{k}{\tau^3} N^{2\tau} \log N$ original points and at most k level-1 medians. All other buckets are formed by combination of other buckets and do not retain original points. They contain $O(\frac{1}{\tau} N^\tau)$ higher level medians and as explained above, each such median requires at most $O(k^{\frac{1}{\tau}} \log N)$ memory for estimating number of active points assigned to it. If $k^{\frac{1}{\tau}} \log N < N^\tau$ then the memory requirement of such a bucket is also $O(\frac{1}{\tau} N^{2\tau})$. This gives a total memory requirement of $O(\frac{k}{\tau^4} N^{2\tau} \log^2 N)$. We have already established the approximation factor for this algorithm in Lemma 6. This gives us our final result which is summarized in the theorem below.

THEOREM 3. *For $\tau < 1/2$, the SKWM algorithm provides a $2^{O(1/\tau)}$-approximation. It uses $O(\frac{k}{\tau^4} N^{2\tau} \log^2 N)$ memory and requires $\tilde{O}(k)$ amortized maintenance time per data element, provided $k^{\frac{1}{\tau}} \log N < N^\tau$.*

4. CONCLUSIONS AND OPEN PROBLEMS

The goal of algorithms for data stream processing under the sliding window model is to maintain statistics or information for the most recent N members of a point set that is growing in real time, while operating with memory that is asymptotically smaller than the window size. We have introduced two such algorithms. The first uses $O(\frac{1}{\epsilon^2}\log N)$ memory and maintains an estimate of the variance of the most recent N real numbers from a growing stream, with at most ϵ relative error (where ϵ is between 0 and 1). Closing the gap between this upper bound on memory usage, and the lower bound of $\Omega(\frac{1}{\epsilon}\log N)$, given by Datar et al.[7], is an open problem. The second algorithm uses $O(\frac{k}{\tau^4}N^{2\tau}\log^2 N)$ memory, for τ between 0 and 1/2, and maintains a solution to the k–median problem for the most recent N points in a stream. This algorithm maintains k medians whose cost on the most recent N points is always within a constant factor of the cost of the optimal k medians for these N points. Whether it is possible to maintain approximately optimal medians in polylogarithmic space (as Charikar et al. [6] do in the stream model without sliding windows), rather than polynomial space, is an open problem.

5. REFERENCES

[1] V. Arya, N. Garg, R. Khandekar, V. Pandit, A. Meyerson, and K. Munagala. "Local Search Heuristics for k–median and Facility Location Problems." In *Proc. 33rd ACM Symp. on Theory of Computing (STOC)*, 2001, pages 21–29.

[2] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. "Models and Issues in Data Stream Systems." In *Proceedings of the 21st ACM Symposium on Principles of Databases Systems (PODS)*, 2002, pages 1–16.

[3] B. Babcock, M. Datar, and R. Motwani. "Sampling from a Moving Window over Streaming Data." In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2002, pages 633–634.

[4] P. S. Bradley, U. M. Fayyad, and C. Reina. "Scaling clustering algorithms to large databases." In *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining (KDD)*, 1998, pages 9–15.

[5] M. Charikar and S. Guha. "Improved Combinatorial Algorithms for the Facility Location and k–median Problems." In *Proc. of the 40th Annual IEEE Symposium on Foundations of Computer Science(FOCS)*, 1999, pages 378–388.

[6] M. Charikar, L. O'Callaghan, and R. Panigrahy. "Better Streaming Algorithms for Clustering Problems." In *Proc. of 35th ACM Symposium on Theory of Computing (STOC)*, 2003.

[7] M. Datar, A. Gionis, P. Indyk, and R. Motwani. "Maintaining Stream Statistics over Sliding Windows." In *Proceedings of Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2002, pages 635–644.

[8] P. Domingos and G. Hulten. "Mining High-speed Data Streams." In *Proceedings of the 6th International Conference on Knowledge Discovery and Data Mining (KDD)*, 2000, pages 71–80.

[9] P. Domingos, G. Hulten, and L. Spencer. "Mining Time-changing Data Streams." In *Proceedings of the 7th International Conference on Knowledge Discovery and Data Mining (KDD)*, 2001, pages 97–106.

[10] P. Gibbons, and S. Tirthapura. "Distributed Streams Algorithms for Sliding Windows" In *Proc. of ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2002.

[11] A. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss. "Surfing Wavelets on Streams: One-pass Summaries for Approximate Aggregate Queries." In *Proc. 27th Conf. on Very Large Data Bases (VLDB)*, 2001, pages 79–88.

[12] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. "Clustering Data Streams." In *Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2000, pages 359–366.

[13] S. Guha, R. Rastogi, and K. Shim. "CURE: An efficient clustering algorithm for large databases." In *Proc. of the ACM SIGMOD Intl. Conference on Management of Data (SIGMOD)* , 1998, pages 73–84.

[14] M.R. Henzinger, P. Raghavan, and S. Rajagopalan "Computing on Data Streams.", Technical Report 1998-011, Compaq Systems Research Center, Palo Alto, CA, May, 1998.

[15] P. Indyk. "Sublinear Time Algorithms for Metric Space Problems." In *Proceedings of the 31st Annual ACM Symposium on Theory of Computing (STOC)*, 1999, pages 428–434.

[16] K. Jain and V. Vazirani. "Primal-Dual Approximation Algorithms for Metric Facility Location and k–median Problems." In *Proc. 40th IEEE Symp. on Foundations of Computer Science (FOCS)*, 1999, pages 1–10.

[17] R. R. Mettu and C. G. Plaxton. "The Online k–median Problem." In *Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2000, pages 339-348.

[18] N. Mishra, D. Oblinger, and L. Pitt. "Sublinear Time Approximate Clustering." In *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2001, pages 439–447.

[19] L. O'Callaghan, N. Mishra, A. Meyerson, S. Guha, and R. Motwani. "Streaming-Data Algorithms for High-Quality Clustering." In *Proceedings of the 18th Annual IEEE International Conference on Data Engineering (ICDE)*, 2001.

[20] C. R. Palmer and C. Faloutsos. "Density biased sampling: an improved method for data mining and clustering." In *Proc. of the ACM SIGMOD Intl. Conference on Management of Data (SIGMOD)*, 2000, pages 82–92.

[21] D. Pelleg and A. W. Moore. "Accelerating exact k–means algorithms with geometric reasoning." In *Proceedings of the 5th International Conference on Knowledge Discovery and Data Mining (KDD)*, 1999.

[22] T. Zhang, R. Ramakrishnan, and M. Livny. "BIRCH: an efficient data clustering method for very large databases." In *Proc. of the ACM SIGMOD Intl. Conference on Management of Data (SIGMOD)*, 1996, pages 103–114.