Computing Diameter in the Streaming and Sliding-Window Models[1]

Joan Feigenbaum[2 5] Sampath Kannan[3 5] Jian Zhang[4 5]

Abstract

We investigate the diameter problem in the *streaming* and *sliding-window* models. We show that, for a stream of n points or a sliding window of size n, any exact algorithm for diameter requires $\Omega(n)$ bits of space. We present a simple ϵ-approximation[6] algorithm for computing the diameter in the streaming model. Our main result is an ϵ-approximation algorithm that maintains the diameter in two dimensions in the sliding-window model using $O(\frac{1}{\epsilon^{3/2}} \log^3 n(\log R + \log\log n + \log \frac{1}{\epsilon}))$ bits of space, where R is the maximum, over all windows, of the ratio of the diameter to the minimum non-zero distance between any two points in the window.

Keywords: Massive Data Streams, Sliding Window, Diameter

---

[1]This work was supported by the DoD University Research Initiative (URI) administered by the Office of Naval Research under Grant N00014-01-1-0795.

[2]Dept. Computer Science, Yale University, P.O. Box 208285 New Haven, CT 06520-8285, USA; Tel: +1-203-432-6432; Fax: +1-203-432-0593;
email: feigenbaum-joan@cs.yale.edu. Supported in part by ONR grant N00014-01-1-0795 and NSF grants CCR-0105337 and ITR-0331548.

[3]Dept. Computer and Information Science, University of Pennsylvania, Philadelphia, PA 19104, USA; Tel: +1-215-898-9514; Fax: +1-215-898-0587;
email: kannan@cis.upenn.edu. Supported in part by ARO grant DAAD 19-01-1-0473 and NSF grant CCR-0105337.

[4]Dept. Computer Science, Yale University P.O. Box 208285 New Haven, CT 06520-8285, USA; Tel: +1-203-432-1227; Fax: +1-203-432-0593; email: zhang-jian@cs.yale.edu. Supported by ONR grant N00014-01-1-0795 and NSF grants CCR-0105337 and ITR-0331548.

[5]Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

[6]Denote by A the output of an algorithm and by T the value of the function that the algorithm wants to compute. We say A ϵ-approximates T if $1 > \epsilon > 0$ and $(1+\epsilon)T \geq A \geq (1-\epsilon)T$.

1 Introduction

In recent years, massive data sets have become increasingly important in a wide range of applications. In many of these applications, the input can be viewed as a *data stream* [3, 19, 9] that the algorithm reads in one pass. The algorithm should take little time to process each data element and should use little space in comparison to the input size.

In some scenarios, the input stream may be infinite, and the application may only care about recent data. In this case, the *sliding-window model* [8] may be more appropriate. As in the streaming model, a sliding-window algorithm should go through the input stream once, and there is not enough storage space for all the data, even for the data in one window.

In this paper, we investigate the two-dimensional diameter problem in these two models. Given a set of points P, the diameter is the maximum, over all pairs x, y in P, of the distance between x and y. There are efficient algorithms to compute the exact diameter [6, 25] or to approximate the diameter [1, 2, 4, 5]. However, little has been done for computational-geometry problems in the streaming or sliding-window models. In particular, little is known about the diameter problem in these two models.

Geometric data streams arise naturally in applications that involve monitoring or tracking entities carrying geographic information. For example, Korn *et al.* [24] discuss applications such as decision-support systems for wireless-telephony access. The customers using the service generate streams of data about their locations, and the cell-phone company may want to process and query the streams for various decision-support purposes. Data streams generated by sensor nets and other observatory facilities provide another example [7]. In these applications, the data streams consist of the records of geographically dispersed events, and computing extent measures (such as the diameter) is an important component of monitoring the spatial extent of these events.

In this work, we show that computing the exact diameter for a set of n points in the streaming model or maintaining it in the sliding-window model (with window width n) requires $\Omega(n)$ bits of space. However, when approximation is allowed, we present a simple ϵ-approximation algorithm in the streaming model that uses $O(1/\epsilon)$ space and processes each point in $O(\log(1/\epsilon))$ time. We also present an approximate sliding-window algorithm to maintain the diameter in 2-d using $O(\frac{1}{\epsilon^{3/2}} \log^3 n(\log R + \log\log n + \log \frac{1}{\epsilon}))$ bits of space, where R is the maximum, over all windows, of the ratio of the diameter to the minimum non-zero distance between any two points in the window. All of our algorithms make only one-sided error. That is, the output $\hat{D}$ of our algorithms ϵ-approximates the true diameter D in that $D \geq \hat{D} \geq (1-\epsilon)D$.

2 Models and Related Work

The streaming model was introduced in [3, 19, 9]. A *data stream* is a sequence of data elements $a_1, a_2, \ldots, a_n$. We denote by n the number of data elements in the stream. In this paper, the data elements are points in two-dimensional Euclidean space.

A *streaming algorithm* is an algorithm that computes some function over a data stream and has the following properties:

1. The input data are accessed in sequential order.

2. The order of the data elements in the stream is not controlled by the algorithm.

The sliding-window model was introduced in [8]. In this model, one is only interested in the n most recent data elements. Suppose a_i is the current data element. The window then consists of elements $\{a_{i-n+1}, a_{i-n+2}, \ldots, a_i\}$. When new elements arrive, old elements are aged out. A *sliding-window algorithm* is an algorithm that computes some function over the window for each time instant. Note that the window is a contiguous subset of the data elements in the stream. Properties (1) and (2) above hold in the sliding-window model as well as the streaming model.

Because n (the stream length in the streaming model or the window width in the sliding window model) is large, we are interested in sub-linear space algorithms, especially those using $polylog(n)$ bits of space.

Previous work in the streaming model focuses on computing statistics over the stream. There are streaming algorithms for estimating the number of distinct elements in a stream [11] and for approximating frequency moments [3]. Work has been done on approximating L^p differences or L^p norms of data streams [9, 12, 21]. There are also algorithms to compute histograms [17, 14] and wavelet approximations [15] for the data elements in a stream. Computing quantiles has been studied [16] in the streaming model, too. Previous work in the sliding-window model [8] addresses the maintenance of the sum of the data elements in the window. The same work also shows how to maintain L^p norms in the window. Later, Gibbons *et al.* [13] improved the counting results of [8] and extended the sliding-window model to distributed streams.

However, at the time of this work, aside from the stream-clustering algorithm in [18][7] and the study of the reverse-nearest-neighbor problem in [24], little is known about computational-geometry problems in the streaming or sliding-window models. Note that a streaming or sliding-window version of a computational-geometry problem is a dynamic problem, *i.e.*, the set of geometric objects under consideration may change. There are additions and deletions to the current set of geometric objects as the computation proceeds. However, the difference between a dynamic computational-geometry algorithm and a streaming/sliding-window geometry algorithm is that, in the streaming/sliding-window model, the algorithm only has limited space and cannot store all the geometric objects in its memory. Thus, most dynamic computational-geometry algorithms cannot be applied directly in the streaming/sliding-window model. The study of stream algorithms for basic problems in computational geometry is an interesting new research area.

Agarwal *et al.* [1] presented an algorithm that maintains approximate extent measures of moving points. Among these extent measures is the diameter of the set of the points. In [1], this algorithm is not presented as a streaming algorithm. However, it can be simply adapted to compute the diameter in the streaming model. Our streaming computation of the diameter takes a different approach, reducing the amount of time required to process

[7] Note that the stream-clustering algorithm can be applied in any metric space, not just in the Euclidean space.

each point while using more space. Note that the moving-points model and the sliding-window model are different and thus that the results for maintaining the diameter in the moving-points model in [1] are not directly comparable to our results in the sliding-window model.

A preliminary version of this work was presented in [10]. Since then, more work has been done on geometric problems in the streaming model. In [20], Hershberger *et al.* provide a streaming algorithm that ϵ-approximates the diameter using $O(\frac{1}{\sqrt{\epsilon}})$ space and $O(\log(\frac{1}{\epsilon}))$ time per point, thus improving the space upper bound given by our approximation. The heart of their algorithm is the computation of the convex hull of the adaptively sampled extrema. The same convex hull is also used in [20] to approximate other properties, such as enclosure and linear separation, of a stream of points. Using a technique similar to ours, Cormode *et al.* [7] devised the radial histogram to estimate the diameter, furthest neighbor, and convex hull of a stream of points. Furthermore, spatial joints and spatial aggregation such as reverse-nearest neighbors can be estimated using multiple radial histograms.

Although all of the above work addresses computational-geometry problems in low dimensions in the streaming model, we remark that the set of problems in high dimensions are also important. For example, there are problems in information retrieval that are modeled as high-dimensional computational-geometry problems. A streaming algorithm presented in [22] c-approximates the diameter (for $c > \sqrt{2}$) in a d-dimensional space using $O(dn^{1/c^2-1})$ space. However, most high-dimensional geometric problems remain open in the streaming model.

3 Sector-Based Diameter Approximation in the Streaming Model

In this section, we provide a streaming algorithm for approximating the diameter of a set of points in the plane.

A simple streaming adaptation of the algorithm of [1] gives an ϵ-approximation of the diameter using $O(1/\sqrt{\epsilon})$ space and time. Let l be a line and $p, q \in P$ be two points that realize the diameter. Denote by $\pi_l(p), \pi_l(q)$ the projection of p, q on l. Clearly, if the angle θ between l and the line pq is smaller than $\sqrt{2\epsilon}$, $|\pi_l(p)\pi_l(q)| \geq |pq| \cos\theta \geq (1 - \frac{\theta^2}{2})|pq| \geq (1-\epsilon)|pq|$. By using a set of lines such that the angle between pq and one of the lines is smaller than $\sqrt{2\epsilon}$, the algorithm can approximate the diameter with bounded error. The adaptation can go through the input in one pass, project the points onto each line, and maintain the extreme points for the lines. Thus, it is a streaming algorithm.

However, the time taken per point by this algorithm is proportional to the number of lines used, which is $\Omega(1/\sqrt{\epsilon})$. We now present an almost equally simple algorithm that reduces the running time to $O(\log(1/\epsilon))$. Our basic idea is to divide the plane into sectors and compute the diameter of P using the information in each sector. Sectors are constructed by designating a point x_0 as the center and dividing the plane using an angle of θ. We show two sectors in Figure 1.

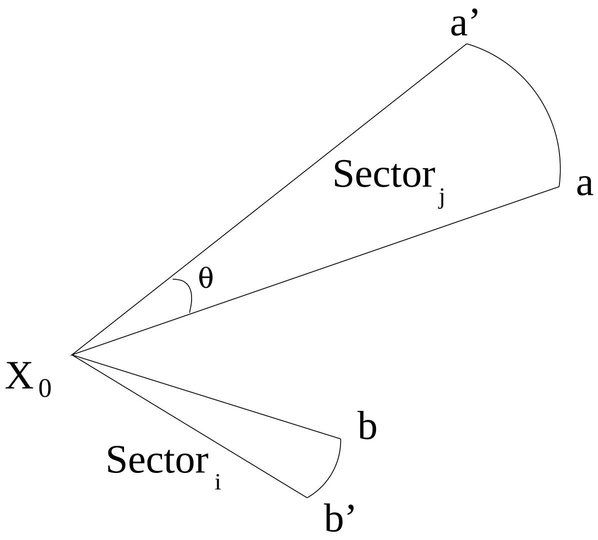

Figure 1: Two Examples of Sectors

Algorithm Streaming-Diameter

1. Take the first point of the stream as the center, and divide the plane into sectors according to an angle $\theta = \frac{\epsilon}{2(1-\epsilon)}$, where ϵ is the error bound. Let S be the set of sectors.
2. While going through the stream, for each sector, record the point in that sector that is the furthest from the center. Also keep track of the maximum distance, R_c, between the center and any other point in P.
3. Let $|ab|$ be the distance between points a and b. Define $D_{max}^{ij} = \max |uv|$ for $u \in$ boundary arc of sector i and $v \in$ boundary arc of sector j, and define $D_{min}^{ij} = \min |uv|$ for $u \in$ boundary arc of sector i and $v \in$ boundary arc of sector j. Output $\max\{R_c, \max_{i,j\in S} D_{min}^{ij}\}$ as the diameter of the point set P.

The sectors have outer boundaries (the arcs aa' and bb' in the figure) that are determined by the distance between the center and the farthest point from the center in that sector. The algorithm records the farthest point for each sector while it goes through the input stream. The full description of the algorithm is given in algorithm "Streaming-Diameter." The algorithm's space complexity is determined by the sector angle θ.

Claim 1. *The distance between any two points in sector i and sector j is no larger than* $\max\{R_c, D_{max}^{ij}\}$. *(Here i could be equal to j.)*

Proof. Consider sectors in figure 1. Let u be a point in sector i and v be a point in sector j. Extend x_0u until it reaches the arc aa'. Denote the intersection point u'. Also extend x_0v until it reaches the arc bb'. Denote the intersection point v'. Then we have $|uv| \leq \max\{|x_0v|, |vu'|\} \leq$

$\max\{R_c, |x_0u'|, |u'v'|\} \leq \max\{R_c, D^{ij}_{max}\}$. (In the two inequalities above we have used (twice) the fact that, if a, b, c occur in that order on a line and d is some point, then $|db| \leq \max(|da|, |dc|)$. $\square$

Claim 2. *With notation as in Figure 1 and in the description of the algorithm,* $D^{ij}_{max} \leq D^{ij}_{min} + length(aa') + length(bb') \leq D^{ij}_{min} + 2R_c \cdot \theta$.

Proof. Consider sectors in Figure 1. Let $|uv| = D^{ij}_{max}$ and $|u'v'| = D^{ij}_{min}$. Because $u, u' \in arc\ aa'$ and $v, v' \in arc\ bb'$, there is a path from u to v, namely $u \sim u' \sim v' \sim v$. Therefore $D^{ij}_{max} \leq |uu'| + D^{ij}_{min} + |vv'| \leq D^{ij}_{min} + 2R_c \cdot \theta$. $\square$

Assume that the true diameter $diam_{true}$ is the distance between a point in sector i and another point in sector j. Let $diam$ be the diameter computed by our algorithm. We observe the following:

$$\max\{R_c, D^{ij}_{min}\} \leq \max\{R_c, \max_{m,n \in S} D^{mn}_{min}\} = diam \leq diam_{true} \leq \max\{R_c, D^{ij}_{max}\}$$

Depending on the relationship between R_c and D^{ij}_{min}, we consider two cases: In the case where $R_c \geq D^{ij}_{min}$, we want $R_c \geq (1-\epsilon)D^{ij}_{max}$ in order to bound the error. This leads to $\theta \leq \frac{\epsilon}{2(1-\epsilon)}$. In the case where $R_c < D^{ij}_{min}$, we want $D^{ij}_{min} \geq (1-\epsilon)D^{ij}_{max}$. Again, this leads to $\theta \leq \frac{\epsilon}{2(1-\epsilon)}$. We will have $O(\frac{1}{\epsilon})$ sectors. Given the center and another point, it takes $O(\log(1/\epsilon))$ time to identify the sector where the point is located. We then have the following theorem.

Theorem 1. *There is an algorithm that ϵ-approximates the 2-d diameter in the streaming model using storage for $O(\frac{1}{\epsilon})$ points. In order to process each point, it takes $O(\log(1/\epsilon))$ time.*

The above algorithm does not work in the sliding-window model. In the streaming model, the boundaries of sectors only expand. This nice property allows us to keep only the extreme points. However, in the sliding-window model, the diameter may decrease with different windows. One may need more information in order to report the diameter for each window.

4 Maintaining the Diameter in the Sliding-Window Model

In this section, we present a sliding-window algorithm that maintains the diameter of a set of points in the plane. Note that by applying the projection technique used in the diameter algorithm of [1], we can transform the problem of maintaining the diameter in two-dimensional Euclidean space to the problem of maintaining the diameter in one-dimensional space. Hence, we first consider the problem on a line.

In the sliding-window model, each point has an age indicating its location in the current window. We call the recently arrived points *new* points and the expiring points *old* points. We denote by $|ab|$ the distance between point a and point b. We also say that the distance $r = |ab|$ is *realized* by points a and b. We may further say that r is realized by a when it is not necessary to mention b or when b is clear in context. In particular, we denote by $diam_a$ the diameter realized by a.

Our main algorithm employs a subroutine that we call the *rounding* subroutine. When invoked on a set of points, the rounding subroutine produces a space-efficient representation for the set of points. We call this representation a *cluster*. Once the cluster is constructed, the diameter of the set of points can be approximated by computing the diameter of this cluster. We now describe the rounding subroutine in detail. (Note that, for simplicity, we describe here the rounding subroutine invoked on one set of points. In our main algorithm that maintains the diameter in the sliding-window model, the rounding subroutine will be invoked on different subsets of points in the window, as well as on the clusters, which essentially are also sets of points.)

Given three points a, b, c and an approximation error of $\hat{\epsilon}$, if we treat point c as a center (the coordinate zero), we can "round" point b to point a if $|ac| \leq |bc| \leq (1+\hat{\epsilon})|ac|$. For a set of points, we can pick some point in the set as the center and round the other points in the same manner. Let c be the point in the set picked as the center. Let d be the minimum distance between c and any other point in the set. Consider the distance intervals $[c, t_0), [t_0, t_1), [t_1, t_2), \ldots, [t_{k-1}, t_k]$, such that $|ct_i| = (1+\hat{\epsilon})^i d$. The rounding subroutine rounds down (moves) each point in the interval $[t_i, t_{i+1})$ to the location t_i (Figure 2).

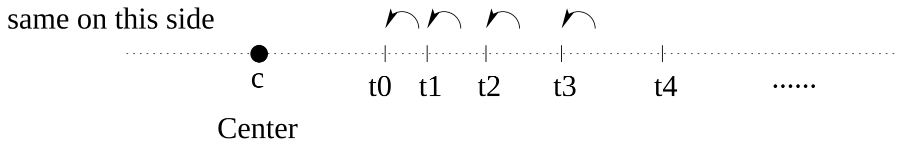

Figure 2: Rounding Points in Each Interval

If multiple points are rounded to the same location, the subroutine discards the older ones and keeps only the newest one. There is at most one point in each of these intervals. The cluster for this set of points consists of the center c and the points that are kept by the subroutine after rounding them. Note that the points in a cluster are different from the points in the original set. We say a point a in the cluster *represents* a point b in the original set if b is rounded to the location of a by the rounding subroutine, regardless of whether b is discarded. (We call the point a the *representative point* of the *original* point b.) Note that a point a in the cluster may represent several points in the original set that are rounded to its location. The location of the representative point in the cluster may not be the same as the location of the point in the original set.

The *volume* of a cluster is the number of the points in the original set that are represented by the cluster. We say a cluster is at level ℓ if the volume of the cluster is 2^ℓ. The *size* of a cluster is the number of points in the cluster. Let D be the diameter of a set of points. The size of the cluster constructed by invoking the rounding subroutine on this set is at most:

$$k \leq 2\log_{1+\hat{\epsilon}} \frac{D}{d} = 2\frac{\log D/d}{\log e \ln(1+\hat{\epsilon})} \leq \frac{4}{\hat{\epsilon}\log e}\log\frac{D}{d}$$

Hence, given a set of points with diameter D and an arbitary point c in the set, if the non-zero minimal distance between c and any other point in the set is d and if the approximation allows an error of $\hat{\epsilon}$, the rounding subroutine constructs a cluster. The diameter of the set of points can be approximated by the diameter of this cluster. The size of the cluster is upper bounded by

$\frac{4}{\hat{\epsilon}\log e}\log\frac{D}{d}$. Also, we observe that there may be a displacement between an original point in the set and its representative point in the cluster. In this case, the representative point is always closer to the center than the original point. Thus, we say that the point in the original set is rounded (moved) "toward the center".

Claim 3. *When invoked on a set of points, the rounding subroutine constructs a cluster that can be used to approximate the diameter of the set of points. If there is a displacement between a point in the set and its representative point in the cluster, the point will only be rounded (moved) toward the center of the cluster.*

We now proceed to describe our main algorithm that maintains the diameter in the sliding-window model. A cluster constructed for a set of points may not be useful for approximating the diameter if, after the construction of the cluster, some of the points in the set have been deleted and some new points have been added. For a static set of points, by constructing the cluster, we are able to represent a point, say b, by another point, say a, because there is some distance (for example $|bc|$) realized by b that promises a lower bound for any diameter that may be realized by b, and the displacement (if any) because of rounding is at most $\hat{\epsilon}|bc|$. In the sliding-window model, such a promise provided by the point c may be broken when c expires. On the other hand, a cluster is essentially a set of points too. If we have a cluster that represents all the points in the current window, each time a new point arrives, we may delete the expired point from the cluster and view the new point and the remaining points in the cluster as a new set of points. A center can be selected, and the rounding subroutine can be applied on this set to construct a new cluster. However, this approach will result in too many roundings and introduce too much error in the approximation.

To overcome these problems, we maintain multiple clusters with the following properties:

1. A cluster represents an interval of points in the window (the continuous subsequence of points within a time interval of the window). The newest point in the interval is picked to be the center. The rounding subroutine is invoked on this interval of points, and the resulting cluster represents these points in the window interval.

2. The clusters are at level $1, 2, \ldots, \lfloor \log n \rfloor$.

3. We allow at most two clusters at each level.

4. When the number of clusters at level i exceeds 2, the oldest two clusters (where the age of a cluster is determined by the age of its center) at that level are merged to form a cluster at level $i + 1$.

Imagine a tree built on the original input points in a window. The points are the leaves. Two consecutive points can form a node (a cluster) at level 1. Two consecutive level-1 clusters can merge to form a node (a cluster) of level 2. This can be repeated recursively until we reach the top level. In this structure, the original input points represented by a cluster are the leaves of the subtree rooted at the node corresponding to that cluster. Note that, at each level, we only keep at most 2 nodes (clusters). The original input points represented by all the clusters that we keep form a cover of the window. Thus, the whole window can be represented by $O(\log n)$ clusters. Figure 3 shows an example of the clusters built on a window.

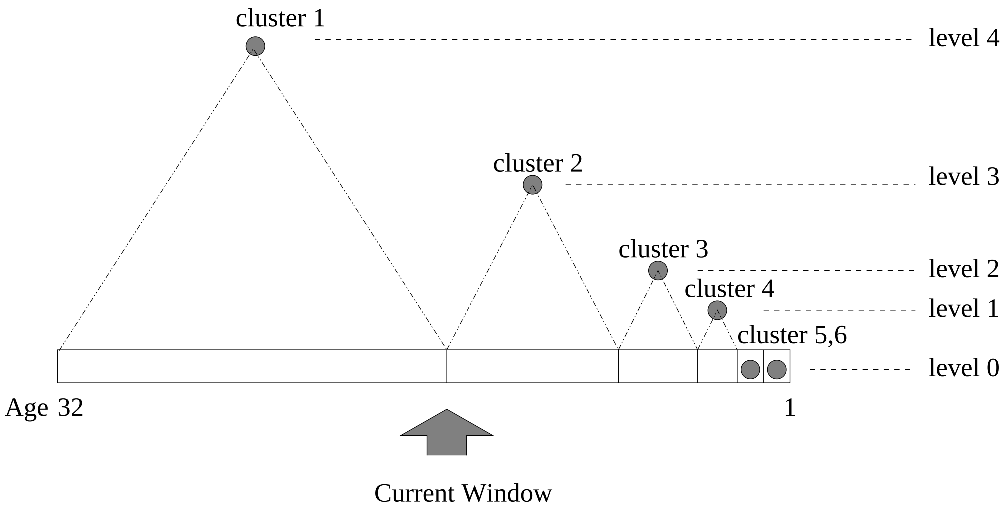

Figure 3: Clusters built for the First Window

When the window slides forward, new points are added to the window and new clusters are formed. To maintain the required number of clusters at each level, clusters are merged whenever there are too many clusters at some level. Once a cluster reaches the top level, it stays at that level. Points in this cluster will ultimately be aged out until the whole cluster is gone.

In order to merge clusters c_1 centered at Ctr_1 and c_2 centered at Ctr_2 to form cluster c_3, we go through the following steps: (We can assume, *w.l.o.g.* c_2 is newer than c_1.),

1. Use Ctr_2 as the center of newly formed cluster c_3.

2. Discard the points in c_1 that are located between the centers of c_1 and c_2.

3. After step (2), if any point p in c_1 satisfies $|pCtr_2| < |pCtr_1| \leq (1 + \hat{\epsilon})|Ctr_1Ctr_2|$, discard p.

4. Let P_{merge} consist of the remaining points of c_1 and the points in c_2. Invoke the rounding subroutine on P_{merge} with Ctr_2 as the center. Note that the new value of d is the non-zero minimal distance from Ctr_2 to any other point in P_{merge}. It may be different from the one used in building the cluster c_2. The rounding subroutine may round down the points in cluster c_2 too.

From step (4), we know that the new value of d is the non-zero minimal distance between Ctr_2 and any other point in P_{merge}. Let p_{min} be this minimal distance point. If p_{min} belongs to cluster c_1, the distance $|Ctr_2p_{min}|$ may be much smaller than the distance between the point Ctr_2 and the original point(s) represented by p_{min}. This happens because when the points are rounded to form the cluster c_1, the rounding is based on the distance between these points and the center Ctr_1, not the point Ctr_2. Thus we can't lower bound the value of d for the new cluster c_3 by the minimal distance between its center and any other original point whose representative point is in the cluster. However, step (2) and (3) assure that $|Ctr_2 p_{min}|$ is at least $\hat{\epsilon} \cdot |Ctr_1 Ctr_2|$. Otherwise, p_{min} will be discarded. We know that the two points Ctr_1 and Ctr_2 are at their original locations. Thus, d is bounded by $\hat{\epsilon}$ times the minimal distance between the cluster center and any other original points whose representative point is in the cluster. The lower bound for the whole window will then be the minimum over all the clusters.

In both the rounding subroutine and the merging process, we may discard points. We now show that if any of the discarded points has the potential of realizing the diameter, it is represented by some representative point in the resulting cluster.

Lemma 1. *In the rounding subroutine or in the merging process, if a point is discarded, either it will not realize any diameter or it is represented (by some representative point) in the cluster resulting from the rounding subroutine or the merging process.*

Proof. The case with the rounding subroutine is clear because a point b is discarded only if there is already a representative point a in the cluster for b and the age of a is smaller than that of b.

In the merging process, we discard two types of points. For two clusters c_1 and c_2 with centers Ctr_1 and Ctr_2, the first type of points we discard are the points in c_1 that are located between Ctr_1 and Ctr_2. Let p_0 be such a point. Note that for any distance realized by p_0 and some other points p, there is a point $p' \in \{Ctr_1, Ctr_2\}$ such $|pp'| > |pp_0|$. Hence, the first type of points we discard will not realize any diameter. The second type of points we discard are the set of points $S = \{p : p \in c_1 \text{ and } |pCtr_2| < |pCtr_1| \leq (1+\hat{\epsilon})|Ctr_1Ctr_2|\}$. For the points in S, Ctr_2 is their representative point in the new cluster resulting from the merging process. $\square$

Define a *boundary point* in a cluster to be an extreme point. (Because here the points are on a line, the extreme point is the point having the largest(smallest) coordinate.) We keep track of the boundary points for each cluster as well as the boundary points for the whole window. Points may expire from the oldest cluster, and this may require updating the boundary points of this cluster. The whole process is summarized in algorithm "sliding-window diameter".

Call the time during which an original point is within some sliding window the *lifetime* of that point. Let's trace a point p through its lifetime. For simplicity, in what follows, instead of saying that the original point p is represented by some point in some cluster, we will just say p is contained or included in that cluster. When clusters merge, if the new representative point of p is located at a different location from the location of the old representative point of p, we will just say p has been rounded ("moved") to a new location and there is a displacement between the new and the old locations of p.

Let p_0 be the original location of the point p and Ctr_0 be the center

Algorithm Sliding-Window Diameter

Update: when a new point arrives:

1. Check the age of the boundary points of the oldest cluster. If one of them has expired, remove it and update the boundary point.
2. Make the newly arrived point a cluster of size 1. Go through the clusters from the most recent to the oldest and merge clusters whenever necessary according to the rules stated above. Update the boundary points of the clusters resulting from merges.
3. Update the boundary points of the window if necessary.

Query Answer: Report the distance between the boundary points of the window as the window diameter.

of the first cluster that includes the point p. When this cluster and some other cluster merge, p could be rounded to a new location p_1. Let Ctr_1 be the center of the newly formed cluster. If we continue this process, before p expires or is no longer represented by any representative point in the window, we will have a sequence of p's locations $p_0, p_1, \ldots$ and corresponding sequence of centers $Ctr_0, Ctr_1, \ldots$. Assume at certain time t, p realizes the diameter $diam_p$. Let $p_0, p_1, \ldots, p_t$ be the sequence of p's locations up to the time t and $Ctr_0, Ctr_1, \ldots Ctr_t$ the corresponding sequence of centers. We have the following claim:

Figure 4: Point may be moved in each rounding but all the displacement are in the same direction.

Claim 4. *If a point is rounded multiple times during its lifetime, all the displacement because of rounding are in the same direction(Figure 4). That is, for all the locations p_i and all the corresponding centers Ctr_i, $|p_0Ctr_i| \geq |p_iCtr_i|$. Furthermore, if a point realizes the diameter at certain time, the distance between the original point and any of its cluster centers up to that time is at most the distance of this diameter. That is, for $i = 1, 2, \ldots, t$, $|p_0Ctr_i| \leq diam_p$.*

Proof. Suppose that the first time p is rounded, it is rounded to the right. If now p is rounded to the left for the first time on step i, then by Claim 3, Ctr_{i-1} lies to the right of p while Ctr_i lies to the left. Further, p belonged to the cluster of Ctr_{i-1} before the merge. Hence, by our rules it would have been *discarded*, because it lies between the two centers, and it belongs to the older cluster. Furthermore, as shown in the proof of Lemma 1, such a point p will not realize the diameter and would no longer be represented by any representative point in the clusters in the window.

Note that $diam_p$ is the true diameter realized by the point p. p_0 is the original location of p and for $i = 1, 2, \ldots, t$, the center Ctr_i is also at its original location. Further, these centers are no older than the point p in the window. Hence, for $i = 1, 2, \ldots, t$, the diameter realized by p is at least the distance $|p_0Ctr_i|$. □

We bound the error in the rounding process by showing that, for all i, $|p_0p_i|$ is at most an ϵ fraction of the diameter realized by p.

Each time we round a point, we may introduce some displacement or error. Let $err_{i+1} = |p_ip_{i+1}|$ be the displacement introduced in the $i+1$th merging. We have the following lemma:

Lemma 2. *The total rounding error of a point p, before it expires or is no longer represented by any representative point in the window, is at most $\hat{\epsilon} \log n \cdot diam_p$.*

Proof. We examine the two cases where there may be a displacement between p_i and p_{i+1}. In the rounding case, we maintain $|p_i Ctr_{i+1}| \leq (1 + \hat{\epsilon})|p_{i+1} Ctr_{i+1}|$. Hence, $err_{i+1} = |p_i p_{i+1}| \leq \hat{\epsilon}|p_{i+1} Ctr_{i+1}| \leq \hat{\epsilon}|p_0 Ctr_{i+1}|$, where the last inequality is by Claim 4. The second case is step (3) of the merging process. If p_i satisfies the condition, it will be discarded. As stated in the proof of Lemma 1, in this case, Ctr_{i+1} becomes the new representative point of p. Hence, p_{i+1} is Ctr_{i+1} and $err_{i+1} = |p_i p_{i+1}| \leq \hat{\epsilon}|Ctr_i Ctr_{i+1}| \leq |p_i Ctr_i|$. By Claim 4, $err_{i+1} \leq \hat{\epsilon}|p_0 Ctr_i|$.

A point may participate in at most $\log n$ merges. The total amount of displacement is then at most $\sum_i err_i \leq \hat{\epsilon} \log n \cdot \max_i |p_0 Ctr_i|$. By Claim 4, we also have $diam_p \geq \max_i |p_0 Ctr_i|$. The lemma follows. $\square$

To bound the error by $\frac{1}{2}\epsilon$, we make $\hat{\epsilon} \leq \frac{\epsilon}{2 \log n}$. The number of points in a cluster after rounding will then be $O(\frac{1}{\epsilon} \log n \log \frac{D}{d})$. Note that for each cluster, d is bounded by $\hat{\epsilon}$ times the minimal distance between the center of the cluster and any other original point whose representative point is in the cluster. Denote by R the maximum, over all windows, of the ratio of the diameter to the minimum non-zero distance between any two original points in that window. Then $\log \frac{D}{d} \leq \log R + \log \frac{1}{\hat{\epsilon}} = O(\log R + \log\log n + \log \frac{1}{\epsilon})$. The number of points in a cluster can then be bounded by $O(\frac{1}{\epsilon} \log n (\log R + \log\log n + \log \frac{1}{\epsilon}))$.

Theorem 2. *There is an ϵ-approximation algorithm for maintaining diameter in one dimension in a sliding window of size n, using $O(\frac{1}{\epsilon}\log^3 n(\log R+\log\log n+\log\frac{1}{\epsilon}))$ bits of space, where R is the maximum, over all windows, of the ratio of the diameter to the minimum non-zero distance between any two points in that window. The algorithm answers the diameter query in $O(1)$ time. Each time the window slides forward, the algorithm needs a worst case time of $O(\frac{1}{\epsilon}\log^2 n(\log R+\log\log n+\log\frac{1}{\epsilon}))$ to process the incoming point. With a slight modification, the algorithm can process incoming points with $O(\log n)$ amortized time using $O(\frac{1}{\epsilon}\log^2 n(\log n+\log\log R+\log\frac{1}{\epsilon})(\log R+\log\log n+\log\frac{1}{\epsilon}))$ bits of space.*

Proof. By Lemma 1, any point that may realize the diameter for some window has a representative point in one of the clusters we maintain. We now calculate the error in reporting the diameter using these representative points. Set $\hat{\epsilon} \le \frac{\epsilon}{2\log n}$. By Lemma 2, for a point p and the diameter $diam_p$ realized by p, the displacement between the original location of p and the location of its representative points is at most $\hat{\epsilon}\log n \cdot diam_p \le \frac{\epsilon}{2} diam_p$. Because our algorithm reports the diameter realized by the representative points, such a displacement causes error in our diameter approximation. Note that each of the two representative points that realize the reported diameter may introduce an error at most $\frac{\epsilon}{2}$ of the true diameter. Hence, the diameter reported by our algorithm is at least $(1-\epsilon)$ of the true diameter. Also note that the reported diameter is at most the true diameter. Otherwise, let p and p' be the two original points whose representative points $R(p)$ and $R(p')$ realize the diameter reported by our algorithm. Then

$|pp'| < |R(p)R(p')|$. By Claim 3, the displacement between the location of p (or p') and the location of $R(p)$ (or $R(p')$) is caused by rounding (multiple times) p (or p') toward some center(s). Hence, there exists at least one center c such that $|R(p)c|$ or $|R(p')c|$ is larger than $|R(p)R(p')|$. This contradicts the fact that $|R(p)R(p')|$ is the diameter reported by our algorithm.

We now analyze the time and space requirement of our algorithm. For each cluster, we maintain the following information:

1. The exact location of the center and the exact location of the point closest to (but not located at) the center.

2. The age of all the points.

3. The relative positions of all the points other than the center.

The relative positions of all the point in a cluster can be encoded by a bit vector. We may need $\log n$ bits of space to record the age in the current window for each point. Thus, we need $O(\log n)$ bits for each cluster point except the center. There are at most $O(\frac{1}{\epsilon} \log n(\log R + \log \log n + \log \frac{1}{\epsilon}))$ points in each cluster. The space requirement for storing the information in items (2) and (3) for the whole cluster is then $O(\frac{1}{\epsilon} \log^2 n(\log R + \log \log n + \log \frac{1}{\epsilon}))$. Because we assumed that this space is much larger than the space required to store two points, we can neglect the latter (the space for information in item (1)). Given that there are $O(\log n)$ clusters, the total space requirement will be $O(\frac{1}{\epsilon} \log^3 n(\log R + \log \log n + \log \frac{1}{\epsilon}))$ to maintain the diameter.

In order to report the diameter at any time, we maintain the two boundary points for the window while we maintain the clusters. For each cluster, we only need to look at its boundary points, and thus the process of updating the sliding window's boundary points will only cost $O(\log n)$ time.

However, while updating the clusters, we may face a sequence of cascading merges. In the worst case, we may need to merge $O(\log n)$ clusters with $O(\frac{1}{\epsilon}\log n(\log R + \log\log n + \log\frac{1}{\epsilon}))$ points in each. This requires time $O(\frac{1}{\epsilon}\log^2 n(\log R + \log\log n + \log\frac{1}{\epsilon}))$.

If a bit vector is used to specify the relative locations of the points in a cluster, when we process the cluster during merging we may need to go through the zero entries in the vector . This could be a waste of time if the vector is sparse. We can directly specify the relative location of a point instead. Because there are $O(\frac{1}{\epsilon}\log n(\log R + \log\log n + \log\frac{1}{\epsilon}))$ different locations, we need an additional $O(\log\frac{1}{\epsilon} + \log\log n + \log\log R)$ bits, besides the $O(\log n)$ bits stated above, for each point in a cluster. The space requirement for each point in a cluster will then be $O(\log n + \log\log R + \log\frac{1}{\epsilon})$. With this modification, when merging two clusters, we are free of overhead other than processing the points in the clusters. During a point's lifetime, it will take part in at most $\log n$ merges. Thus, a simple analysis can show that the amortized cost for updating is now only $O(\log n)$. □

To extend the algorithm to 2-d, we can apply the projection technique. We use a set of lines and project the points in the plane onto the lines. We guarantee that, for any pair of points, they will project to a line with angle θ such that $1 - \cos\theta \leq \frac{\epsilon}{2}$. This will require $O(\frac{1}{\sqrt{\epsilon}})$ lines. We then use our diameter-maintenance algorithm on lines to maintain the diameter in the 2-d case.

Theorem 3. *There is an ϵ-approximation algorithm for maintaining diameter in 2-d in a sliding window of size n using $O(\frac{1}{\epsilon^{3/2}}\log^3 n(\log R + \log\log n + \log\frac{1}{\epsilon}))$ bits of space, where R is the maximum, over all windows, of the ratio of the diameter to the minimum non-zero distance between any two points in that window.*

5 Lower Bounds

It is well-known that the set-disjointness problem has linear communication complexity [23] and thus a linear-space lower bound in the streaming model. (These lower bounds apply to randomized protocols and algorithms as well.) One can map the set elements to points on a circle such that the diameter of the circle will be realized if and only if the corresponding element is presented in both sets. This reduction gives the following theorem.

Theorem 4. *Any streaming algorithm that computes the exact diameter of n points, even if each point can be encoded using at most $O(\log n)$ bits, requires $\Omega(n)$ bits of space.*

Proof. We reduce the set-disjointness problem to a diameter problem. The set-disjointness problem is defined as follows: Given a set U of size n and two subsets $x \subseteq U$ and $y \subseteq U$, the function $disj(x, y)$ is defined to be "1" when $x \cap y = \phi$ and "0" otherwise. The corresponding language DISJ is the set $\{(x, y)|x \subseteq U,\ y \subseteq U,\ x \cap y = \phi\}$.

The set-disjointness problem has a linear communication complexity lower bound. Because a streaming algorithm can be easily transferred into a one-round communication protocol, the linear communication complexity lower bound gives a linear space lower bound for set-disjointness problem in the streaming model.

Consider points on a circle in the plane. For a given point p_i, there is exactly one other point on the circle such that the distance between it and p_i is exactly equal to the diameter of the circle. Denote this antipodal point p_i'. The distance between p_i and all other points on the circle is smaller than the distance between p_i and p_i'. We map each element $i \in U$ onto one such antipodal pair. We further make the appearance of one point in the pair correspond to the appearance of the element i in subset x and the appearance of the other point correspond to the element i in y. We will have both points p_i and p_i' only if the element i is in both subsets x and y.

Given an instance (x, y) of DISJ, we construct an instance of the diameter problem according to the above principle. We give an example in Figure 5

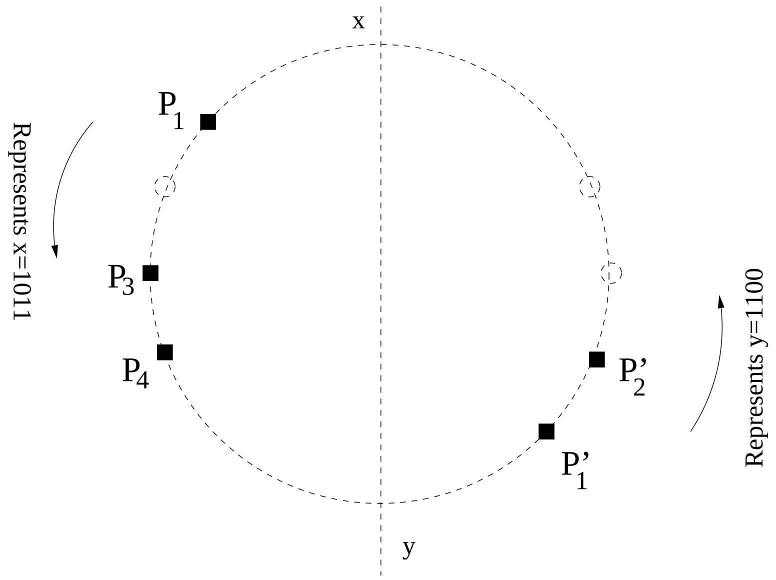

Figure 5: Reduction from DISJ to Diameter

The solid squares in the figure are the points we put into the diameter instance. The DISJ instance in Figure 5 is x, y, where $x = 1011$ and $y = 1100$. The diameter instance contains p_1, p_3, p_4, because $x = 1011$, and p'_1, p'_2, because $y = 1100$. The dashed circles in the figure show the location for p_2, p'_3, p'_4. Because $x_2 = 0$ and $y_3 = y_4 = 0$, these points are not presented in the stream.

In the example, element 1 is in both x and y. The diameter of the point set constructed is $|p_1 p'_1|$ and is exactly the diameter of the circle. On the other hand, if $x \cap y = \phi$, the diameter of the point set will be strictly smaller than the diameter of the circle. Thus, an exact algorithm for the diameter problem could be used to solve the set-disjointness problem. $\square$

We remark that in the above construction, each point in the input stream may be encoded using at most $O(\log n)$ bits. Hence, Theorem 4 gives a space lower bound of $\Omega(n)$ bits for a stream of length $O(n \log n)$ bits.

For a positive number $\epsilon \leq \frac{1}{\pi^2}$, consider the same construction with a set of at most $\frac{1}{\sqrt{\epsilon}}$ points. Let x_ϵ and y_ϵ be the two sets represented by these points and let the diameter of the circle used in the construction be 1. If $x_\epsilon \cap y_\epsilon \neq \phi$, the diameter of the set of points is 1 while if $x_\epsilon \cap y_\epsilon = \phi$, the diameter would be at most $\cos(\pi\sqrt{\epsilon})$. Because $1 - \cos(x) \geq \frac{1}{2}x^2 - \frac{1}{24}x^4 \geq \frac{11}{24}x^2$, $1 - \cos(\pi\sqrt{\epsilon}) \geq \frac{11\pi^2}{24}\epsilon \geq 3\epsilon$, a streaming algorithm that can $(1 \pm \epsilon)$-approximate the diameter will be able to distinguish the two cases and thus solve the set-disjointness problem on the two sets x_ϵ and y_ϵ. Such a streaming algorithm requires $\Omega(\frac{1}{\sqrt{\epsilon}})$ bits of space.

In the sliding-window case, we have a similar bound even for points on a line. Obviously the lower bound holds for higher dimensions as well.

Theorem 5. *To maintain, in a sliding window of size n, the exact diameter of a set of points on a line, even if each point in the set can be encoded using $O(\log n)$ bits, requires $\Omega(n)$ bits of space.*

Proof. Consider a family $\mathscr{F}$ of point sequences of length $2n-2$. Let $\{a_1, a_2, \ldots, a_{2n-2}\}$ be a sequence in $\mathscr{F}$. Because a_i is a point in one dimension, we denote by a_i the point as well as the coordinate (a real number) of the point. The sequences in the family $\mathscr{F}$ have the following properties:

1. For $i = n, n+1, \ldots, 2n-2$, a_i is located at coordinate zero, *i.e.*, $a_i = 0$. Furthermore, $a_{n-1} = 1$.

2. $|a_1 a_n| \geq |a_2 a_{n+1}| \geq |a_3 a_{n+2}| \geq \ldots \geq |a_{n-1} a_{2n-2}|$, *i.e.*, $a_1 \geq a_2 \geq a_3 \geq \ldots \geq a_{n-1}$.

3. The coordinates of the points a_j, for $j = 1, 2, \ldots, n-2$, are picked from the set $\{2, 3, \ldots, n\}$.

For a window that ends at point a_s, the diameter is exactly the distance $|a_s a_{s+n-1}|$. Any two members of the family will have different diameters for a window that ends at a_s, for some $s \in \{1, 2, \ldots, n-2\}$, where the coordinates of a_s differ in the two sequences. Thus, an algorithm that maintains the diameter exactly has to distinguish any two sequences in $\mathscr{F}$.

We need to assign values for the $n-2$ coordinates $a_1 \geq a_2 \geq \ldots \geq a_{n-2}$. By Property (3), we have $n-1$ possible values to choose. (We may assign the same value to multiple coordinates.) The number of such assignments (and thus the number of sequences in $\mathscr{F}$) is $\binom{n-2+n-2}{n-2} \geq 2^{n/2}$. Hence, the algorithm needs $\log|\mathscr{F}| = \Omega(n)$ space. $\square$

We remark that in this family $\mathscr{F}$, the non-zero minimal distance between any two points in a sequence is at least 1. For each sequence, the ratio of the diameter over the non-zero minimal distance between any two points is at most n. Hence, the lower bound holds without the requirement of an extremely large R.

If we change the form of the coordinates of a_j for $j = 1, 2, \ldots, n-2$ to $(1+\epsilon)^{3k}$ while respecting the Property (2) above, a similar family of points sequences can be constructed for ϵ-approximation algorithms. We have the following lower bounds for approximation from this modified family of points sequences.

Theorem 6. *Let R be the maximum, over all windows, of the ratio of the diameter to the minimum non-zero distance between any two points in that window. To ϵ-approximately maintain the diameter of points on a line in a sliding window of size n requires $\Omega(\frac{1}{\epsilon}\log R \log n)$ bits of space if $\log R \leq \frac{3\log e}{2}\epsilon \cdot n^{1-\delta}$, for some constant $\delta < 1$. The approximation requires $\Omega(n)$ bits of space if $\log R \geq \frac{3\log e}{2}\epsilon \cdot n$.*

Proof. Once again consider the family of point sequences in the proof of Theorem 5. We make the following change: The coordinates of the points a_j for $j = 1, 2, \ldots, n-2$, have the form $(1+\epsilon)^{3k}$, for some $k \in \{1, 2, \ldots, \lfloor \frac{1}{3}\log_{(1+\epsilon)} R \rfloor = m\}$. These coordinates are chosen so as to respect Property (2) in the proof of Theorem 5. Note that $\frac{2}{3\log e \cdot \epsilon}\log R \geq m \geq \frac{1}{3\log e \cdot \epsilon}\log R$, for ϵ sufficiently small, because $\epsilon/2 \le \ln(1+\epsilon) \le \epsilon$. Depending on the value of $\log R$, we consider two cases:

1. $\log R \le \frac{3\log e}{2}\epsilon \cdot n^{1-\delta}$ for some constant $\delta < 1$. By a similar argument to the one given in the proof of Theorem 5, the space requirement will now be lower bounded by:

$$\log \binom{m-1+n-2}{m-1} \ge \Omega(m \log \frac{n}{m}) \ge \Omega(\frac{1}{\epsilon} \log R(\delta \log n))$$
$$= \Omega(\frac{1}{\epsilon} \log R \log n)$$

2. $\log R \ge \frac{3\log e}{2}\epsilon \cdot n$. In this case, $m \ge \frac{n}{2}$. We can always choose from $\frac{n}{2}$ different values for the coordinates of the points $a_1, \dots, a_{n-2}$. (Same value can be chosen for multiple coordinates.) The space requirement will be lower bounded by

$$\log \binom{n/2-1+n-2}{n/2-1} \ge \Omega(\frac{n}{2} \cdot \log 2) = \Omega(n)$$

□

6 Conclusion

In this work, we have initiated the study of computational-geometry problems in the streaming and sliding-window models. We provided bounds for approximate and exact diameter computation in these models. Massive streamed data sets for computational-geometry problems arise naturally in applications that involve monitoring or tracking entities carrying geographic information. They are also encountered as problems in areas such as information retrieval and pattern recognition, modeled as computational-geometry problems by means of embedding. Thus, we believe that the study of stream algorithms for basic problems in computational geometry is a promising direction for future research. Indeed, more computational-geometry problems have been studied since our results were first presented in preliminary form, and algorithms and lower bounds are provided in [20, 7, 22]. On the other hand, there are several problems that are still open, among which is the gap between our space upper bound and our lower bound for the diameter problem in the sliding-window model. We conjecture that, for this particular gap, it is the upper bound that can be improved. Another direction that needs further investigation is the window size in the sliding-window model. We use a model that has a fixed window size. In some applications, it is desirable that the window size vary or be approximated, as in the work on elastic windows by Shasha and Zhu [26]. In general, we believe that it would be interesting to study computational-geometry problems in models with variable window size.

References

[1] P. K. Agarwal and S. Har-Peled. Maintaining the approximate extent measures of moving points. In *Proc. 12th ACM-SIAM Symposium on Discrete Algorithms*, pages 148–157, 2001.

[2] P. K. Agarwal, J. Matousek, and S. Suri. Farthest neighbors, maximum spanning trees and related problems in higher dimensions. *Computa-*

tional Geometry: Theory and Applications, 1(4):189–201, 1992.

[3] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, 58(1):137–147, 1999.

[4] G. Barequet and S. Har-Peled. Efficiently approximating the minimum-volume bounding box of a point set in three dimensions. In *Proc. 10th ACM-SIAM Symposium on Discrete Algorithms*, pages 82–91, 1999.

[5] T. Chan. Approximating the diameter, width, smallest enclosing cylinder and minimum-width annulus. In *Proc. 16th ACM Symposium on Computational Geometry*, pages 300–309, 2000.

[6] K. Clarkson and P. Shor. Applications of random sampling in computational geometry II. *Discrete Computational Geometry*, 4:387–421, 1989.

[7] G. Cormode and S. Muthukrishnan. Radial histograms for spatial streams. *DIMACS Tech Report 2003-11*, 2003.

[8] M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining stream statistics over sliding windows. *SIAM Journal on Computing*, 31(6):1794–1813, 2002.

[9] J. Feigenbaum, S. Kannan, M. Strauss, and M. Viswanathan. An approximate L^1 difference algorithm for massive data streams. *SIAM Journal on Computing*, 32(1):131–151, 2002.

[10] J. Feigenbaum, S. Kannan, and J. Zhang. Computing diameter in the streaming and sliding-window models. *Yale University Technical Report, YALEU/DCS/TR-1245*, Dec 2002. Also partially presented at the November 6-9, 2001 and March 24-26 2003 meetings of the DIMACS working-group on streaming data analysis.

[11] P. Flajolet and G. Martin. Probabilistic counting algorithms for data base applications. *Journal of Computer and System Sciences*, 31:182–209, 1985.

[12] J. H. Fong and M. Strauss. An approximate L^p-difference algorithm for massive data streams. *Discrete Mathematics and Theoretical Computer Science*, 4(2):301–322, 2001.

[13] P. B. Gibbons and S. Tirthapura. Distributed streams algorithms for sliding windows. In *Proc. 14th ACM Symposium on Parallel Algorithms and Architectures*, pages 63–72, 2002.

[14] A. C. Gilbert, S. Guha, P. Indyk, Y. Kotidis, S. Muthukrishnan, and M. Strauss. Fast, small-space algorithms for approximate histogram maintenance. In *Proc. 34th ACM Symposium on Theory of Computing*, pages 389–398, 2002.

[15] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss. Surfing wavelets on streams: One-pass summaries for approximate aggregate queries. In *Proc. 27th International Conference on Very Large Data Bases*, pages 79–88, 2001.

[16] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss. How to summarize the universe: Dynamic maintenance of quantiles. In *Proc. 28th International Conference on Very Large Data Bases*, pages 454–465, 2002.

[17] S. Guha, N. Koudas, and K. Shim. Data-streams and histograms. In *Proc. 33th ACM Symposium on Theory of Computing*, pages 471–475, 2001.

[18] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams. In *Proc. 41th IEEE Symposium on Foundations of Computer Science*, pages 359–366, 2000.

[19] M. R. Henzinger, P. Raghavan, and S. Rajagopalan. Computing on data streams. *Technical Report 1998-001, DEC Systems Research Center*, 1998.

[20] J. Hershberger and S. Suri. Convex hulls and related problems in data streams. In *Workshop on Management and Processing of Data Streams*, 2003.

[21] P. Indyk. Stable distributions, pseudorandom generators, embeddings and data stream computation. In *Proc. 41th IEEE Symposium on Foundations of Computer Science*, pages 189–197, 2000.

[22] P. Indyk. Better algorithms for high-dimensional proximity problems via asymetric embeddings. In *Proc. 14th ACM-SIAM Symposium on Discrete Algorithms*, pages 539–545, 2003.

[23] B. Kalyanasundaram and G. Schnitger. The probabilistic communication complexity of set intersection. *SIAM Journal on Discrete Mathematics*, 5:545–557, 1990.

[24] F. Korn, S. Muthukrishnan, and D. Srivastava. Reverse nearest neighbor aggregates over data streams. In *Proc. 28th International Conference on Very Large Data Bases*, pages 814–825, 2002.

[25] E. Ramos. Deterministic algorithms for 3-d diameter and some 2-d lower envelopes. In *Proc. 16th ACM Symposium on Computational Geometry*, pages 290–299, 2000.

[26] Y. Zhu and D. Shasha. Efficient elastic burst detection in data streams. In *Proc. 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 336–345, 2003.