Computing Diameter in the Streaming and Sliding-Window Models*

Joan Feigenbaum†
Dept. of Computer Science
Yale University
feigenbaum-joan@cs.yale.edu

Sampath Kannan ‡
Dept. of Computer & Information Science
University of Pennsylvania
kannan@cis.upenn.edu

Jian Zhang §
Dept. of Computer Science
Yale University
zhang-jian@cs.yale.edu

December 23, 2002

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[1]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 windows 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.

1 introduction

In recent years, massive data sets have become increasingly important in a wide range of applications. In many applications, the input can be viewed as a *data stream* [12, 7] that the

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

†Supported in part by ONR grant N00014-01-1-0795 and NSF grants CCR-0105337, CCR-TC-0208972, ANI-0207399, and ITR-0219018.

‡Supported in part by NSF grant CCR-0105337.

§Supported by NSF grant CCR-0105337.

[1]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 $(1+\epsilon)T \geq A \geq (1-\epsilon)T$.

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* [6] is 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 the 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 [5, 16] or to approximate the diameter [1, 3, 4]. 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.

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 $O(1)$ 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.

The rest of the paper is organized as follows. In Section 2, we briefly introduce the streaming and sliding-window models. In Section 3, we present our streaming diameter-approximation algorithm, and, in Section 4, we present our sliding-window diameter-approximation algorithm. Section 5 shows lower bounds for the exact diameter problem in both models. We also discuss space requirements for approximation in this section. Section 6 concludes this paper.

2 models and related work

The streaming model was introduced in [12, 7]. A *data stream* is a sequence of data elements $a_1, a_2, \ldots, a_n$. We will denote by n the number of data elements in the stream. In this paper, the data elements are points.

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 [6]. 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 subset of contiguous data elements of the stream. Properties (1) and (2) above hold in the sliding-window model as well as the streaming model.

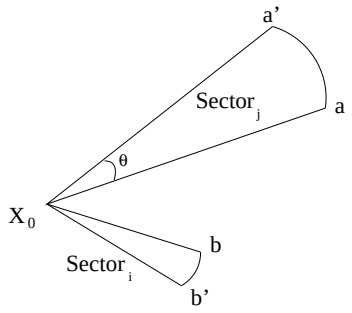

Figure 1: Two Examples of Sectors

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 addresses computing statistics over the stream. There are streaming algorithms for estimating the number of distinct elements in a stream [8] and for approximating frequency moments [2]. Work has also been done on approximating L^p differences or L^p norms of data streams [7, 13]. There are also algorithms to compute histograms for the data elements in the streams [10, 9]. Previous work in the sliding-window model [6] 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. However, aside from the related work on stream clustering [11, 15], little is known about computation-geometry problems in the streaming or sliding-window models.

For the problem of computing the diameter on the plane, the following is a simple algorithm that uses $O(1/\sqrt{(}\epsilon))$ 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 algorithm 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 essentially a streaming algorithm. However, the time taken per point is proportional to the number of lines used, which is $\Omega(1/\sqrt{\epsilon})$. We present an almost equally simple algorithm that removes this dependence of running time on ϵ.

3 A Sector-Based Streaming Diameter-Approximation in the Streaming Model

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.

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 $D^{ij}_{max} = \max |uv|$ for $u \in$ boundary arc of sector i and $v \in$ boundary arc of sector j, and define $D^{ij}_{min} = \min |uv|$ for $u \in$ boundary arc of sector i and $v \in$ boundary arc of sector j. Output $\max\{R_c, \max_{i,j \in S} D^{ij}_{min}\}$ 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 3.1 *The distance between any two points in sector i and sector j is no larger than* $\max\{R_c, D^{ij}_{max}\}$. (*Here i could be equal to j.*)

Proof. 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 the fact that, if a, b, c occur in that order on a line and d is some point, then the $|db| \leq \max(|da|, |dc|)$. ∎

Claim 3.2 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. 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 $D^{ij}_{max} \leq |uu'| + D^{ij}_{min} + |vv'| \leq D^{ij}_{min} + 2R_c \cdot \theta$. ∎

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

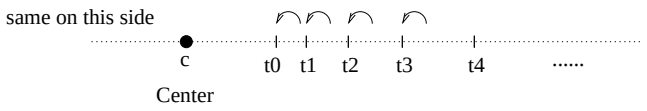

Figure 2: Rounding Points in Each Interval

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 then have the following theorem.

Theorem 3.3 *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 $O(1)$ 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. In next section, we give a deterministic algorithm that maintains an approximation to the diameter in the sliding-window model.

4 maintaining the diameter in the sliding-window model

First, we consider maintaining the diameter for points on a line. In the sliding-window model, each point has an age indicating its location in the current window. The recently arrived points are new points, and the expiring points are 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 b is clear within the context. In particular, the diameter realized by a, denoted $diam_a$, is the maximum distance realized by point a within some window.

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 $|ac| \leq |bc| \leq (1+\hat{\epsilon})|ac|$. Given a set of points in the window, we can pick some point as the center and round the other points in the same manner. We keep the following invariant in rounding:

> **Invariant 4.1** If a point is translocated[2]in a "rounding," it can only be translocated toward the center.

Consider the distance intervals $[c, t_0), [t_0, t_1), [t_1, t_2), \ldots, [t_{k-1}, t_k]$, such that c is the center and $|ct_i| = (1+\hat{\epsilon})^i d$, where d is the minimum distance between the center and any other point. Each point in the interval $[t_i, t_{i+1})$ can be rounded down to t_i (Figure 2).

If multiple points are rounded to the same location, we can discard the older ones and only keep the newest one. We will then have at most one point in each of these intervals.

[2]If a point is rounded to a new location, we say that the point is "translocated".

Let D be the diameter of a set of points. The number k of points that result from rounding all the points in this set will be bounded as follows:

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

We call the set of points that results from rounding a *cluster*. Note that the points in a cluster are different from the points in the original input stream. A point a in the cluster may represent several original points rounded to it. (We call the point a the *representative point* of these original input points in the cluster.) The location of the representative point in the cluster may not be the same as the location of the original points. If 2^l original points are represented by a cluster, the cluster is said to be at *level* l. The diameter of the original set of points can be approximated by the diameter of the cluster.

With this scheme, we are able to round a point, say b, to another point, say a, because there is some distance (for example $|bc|$) realized by b that promises a lower bound for any diameter realized by b, and the error incurred in the rounding is a small fraction of this lower bound. In the sliding-window model, the point c may be aged out in the future, making the approximation error too large. Another issue is whether to recenter the points each time a new point arrives. This will result in too many roundings and introduce too much error in the approximation as well.

To overcome these problems, we maintain multiple clusters each of which has the following properties:

1. A cluster represents an interval of points in the window (the set of points within a time interval of the window). The newest point in the interval is picked to be the center. The other points in the interval are rounded. The resulting points forms a cluster that represents the original points in the window interval.
2. The levels of the clusters are integers.
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.

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

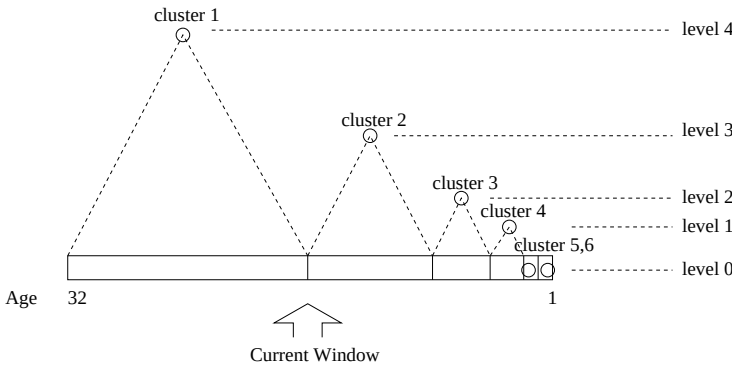

Figure 3: Clusters built for the First Window

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 (2), if point p in c_1 satisfies $|pCtr_2| < |pCtr_1| \leq (1+\hat{\epsilon})|Ctr_1Ctr_2|$, discard p.
4. Let P_{merge} consists of the remaining points of c_1 and the points in c_2. Round points in P_{merge}. The new center is Ctr_2, and the new value of d is the minimum distance from Ctr_2 to any other point in P_{merge}. The new value of d may be different from the one used in building the cluster c_2. We may need to round the points in cluster c_2.

From step (4), we know that the new value of d is the minimum distance between Ctr_2 and any other point in P_{merge}. Let p_{min} be this minimum 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 minimum distance between its center and any other original point whose representative point is in the cluster. However, step (2) (3) assures that $|Ctr_2p_{min}|$ is at least $\hat{\epsilon} \cdot |Ctr_1Ctr_2|$. 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 minimum 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.

Define a *boundary point* in a cluster to be an extreme point. We keep track of the boundary points for each cluster as well as the boundary points for the whole window. Points may expire

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 most recent to 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.

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 life time. 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, instead of saying that the representative point of p is rounded and p has a new representative point in the new cluster, we will just say p is rounded ("translocated") and has a new location now.

Let p_0 be the original location of the point p and Ctr_0 be the center 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 discarded, we will have a sequence of p's locations $p_0, p_1, \ldots, p_t$ and corresponding sequence of centers $Ctr_0, Ctr_1, \ldots, Ctr_t$. We observe that Ctr_i and Ctr_{i+1} will be on the same side of p_i. Otherwise, we would have discarded the point p.

Figure 4: Point may be translocated in each rounding but all the translocations are towards the same direction.

Claim 4.2 *If a point is rounded multiple times during its lifetime, all the translocations because of rounding are in the same direction(Figure 4). In other words, for all the rounded locations p_i and all the corresponding centers Ctr_i, $|p_0 Ctr_i| \geq |p_i Ctr_i|$.*

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 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*, not rounded, because it lies between the two centers, and it belongs to the older cluster. ∎

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.

In an arbitrary rounding scheme, with multiple roundings, a point can be translocated arbitrarily. The distance from the location after rounding to some new center will not promise a lower bound for the diameter realized by the point. However, with our rounding scheme, claim 4.2 guarantees the following invariant:

> **Invariant 4.3** If a point is rounded (even multiple times), the distance between this point after rounding and any of its future cluster centers is at most the distance of any diameter realized by this point.

Each time we round a point, we introduce some dislocation or error. Let $err_{i+1} = |p_ip_{i+1}|$ be the dislocation introduced in the $i+1$th merging. Also, let $diam_p$ be the diameter realized by p in some window. We have the following lemma:

Lemma 4.4 *The total rounding error of point p before it is discarded or expires is at most $\hat{\epsilon}\log n \cdot diam_p$.*

Proof. In each rounding, we maintain $|p_iCtr_{i+1}| \leq (1+\hat{\epsilon})|p_{i+1}Ctr_{i+1}|$. Thus $err_{i+1} = |p_ip_{i+1}| \leq \hat{\epsilon}|p_{i+1}Ctr_{i+1}| \leq \hat{\epsilon}|p_0Ctr_{i+1}|$. A point participates in at most $\log n$ merges. The total amount of translocation is then at most $\sum_i err_i \leq \hat{\epsilon}\log n \cdot \max_i |p_0Ctr_i|$. Also our Invariant 4.3 states that $diam_p \geq \max_i |p_0Ctr_i|$. ∎

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 $O(\frac{1}{\epsilon}\log n \log\frac{D}{d})$. As mentioned above, for each cluster, d is bounded by $\hat{\epsilon}$ times the minimum 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 $\log\frac{D}{d} \leq \log R + \log\frac{1}{\hat{\epsilon}} = \log R + \log\log n + \log\frac{1}{\epsilon}$. 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 4.5 *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. The correctness of our algorithm is clear given the chosen value of $\hat{\epsilon}$ and Lemma 4.4. 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 technique discussed at the beginning of the previous section. We have 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 $1-\cos\theta\le\frac{\epsilon}{2}$. 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 4.6 *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.*

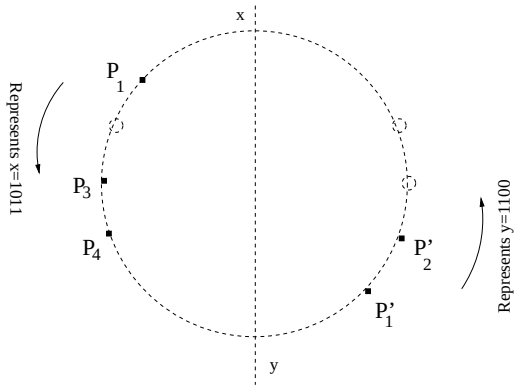

Figure 5: Reduction from DISJ to Diameter

5 lower bounds

It is well-known that the set-disjointness problem has a linear communication complexity [14] and thus a linear space lower bound in the streaming model. 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 two sets. This reduction gives the following theorem.

Theorem 5.1 *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

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.

In the above construction, in order to distinguish the case in which $x \cap y = \phi$ from the case in which $x \cap y \neq \phi$, if the circle has diameter "1," the algorithm must distinguish 1 from $\cos(\frac{\pi}{2n})$. Because $1 - cos(x) \geq \frac{1}{2}x^2 - \frac{1}{24}x^4$, for $x = \frac{\pi}{2n}$ and large n, the difference of $\frac{1}{4n^2}$ must be detectable. This means that the encoding of each point must have precision $\frac{1}{4n^2}$, which can be achieved using $O(\log n)$ bits. ■

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 5.2 *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 $2n-2$. Each sequence $a_1, a_2, \ldots, a_{2n-2} \in \mathscr{F}$ has the following properties:

1. For $i = 1, 2, \ldots, n$, a_{n+i-2} is located at coordinate zero. The coordinate for a_{n-1} is n.
2. $|a_1 a_n| \geq |a_2 a_{n+1}| \geq |a_3 a_{n+2}| \geq \ldots, \geq |a_{n-1} a_{2n-2}|$
3. The coordinates of the points a_j, for $j = 1, 2, \ldots, n-2$, have the form $n \cdot k$ for some $k \in 2, 3, \ldots, n$.

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 $\mathscr{F}$.

By Property (3), the number of member sequences in $\mathscr{F}$ is $\binom{n-2+n-1}{n-1} \geq (1.5)^{n/2}$, for n sufficiently large. (The number of member sequences in $\mathscr{F}$ is in one-to-one correspondence with sequences of 0's and 1's containing $n-2$ 0's and $n-1$ 1's.) The algorithm thus needs $\log |\mathscr{F}| = \Omega(n)$ space. ■

Note that, in this family $\mathscr{F}$, the ratio R is just n. 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 5.3 *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 $\log R \leq \frac{3}{2}\epsilon \cdot n^{1-\delta}$*, for some constant* $\delta < 1$*. The approximation requires* $\Omega(n)$ *bits of space if* $\log R \geq \frac{3}{2}\epsilon \cdot n$*.*

Proof. Once again consider the family of point sequences in the proof of Theorem 5.2. We make the following change: Keep the points $a_n, \ldots, a_{2n-2}$ at coordinate zero, but move the point a_{n-1} to coordinate 1. The coordinates of the points a_j for $j = 1, 2, \ldots, n-2$, have the form $(1+\epsilon)^{3k}$, for some $k \in \{1, 2, \ldots, \frac{1}{3}\log_{(1+\epsilon)} R = m\}$. These coordinates are chosen so as to respect Property (2) in the proof of Theorem 5.2. Note that $\frac{2}{3\epsilon}\log R \geq m \geq \frac{1}{3\epsilon}\log R$, for ϵ sufficiently small, because $\epsilon/2 \leq \log(1+\epsilon) \leq \epsilon$. Depending on the value of $\log R$, we consider two cases:

1. $\log R \leq \frac{3}{2}\epsilon \cdot n^{1-\delta}$ for some constant $\delta < 1$. By a similar argument to the one given in the proof of Theorem 5.2, the space requirement will now be lower bounded by:

$$\begin{aligned}\log \binom{n+m-1}{m} &\geq m \log \frac{n}{m} \geq \frac{1}{3\epsilon}\log R(\delta \log n) \\ &= \Omega(\frac{1}{\epsilon}\log R \log n)\end{aligned}$$

2. $\log R \geq \frac{3}{2}\epsilon \cdot n$. In this case, $m \geq \frac{n}{2}$. We can always choose $\frac{n}{2}$ distinct values for the coordinates of points $a_1, \ldots, a_{n-2}$. The space requirement will be lower bounded by

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

■

6 Future Work

In this paper, we have initiated the study of computational-geometry problems in the streaming and sliding-window models and have provided bounds for approximate and exact diameter computation in these models. Massive streamed data sets for computational-geometry problems arise naturally as problems in areas such as information retrieval and pattern recognition are modeled as computational-geometry problems by means of an embedding into a metric space. Thus, we believe that the study of stream algorithms for basic problems in computational-geometry is a promising direction for future research.

References

[1] P.K. Agarwal, J.Matousek, and S.Suri. Farthest neighbors, maximum spanning trees and related problems in higher dimensions. *Comput. Geom. Theory Appl.*, 1(4):189–201, 1992.

[2] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, 58(1):137–147, Feb. 1999.

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

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

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

[6] M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining stream statistics over sliding windows. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 635–644, 2002.

[7] J. Feigenbaum, S. Kannan, M. Strauss, and M. Viswanathan. An approximate L^1 difference algorithm for massive data streams. In *IEEE Symposium on Foundation of Computer Science*, pages 501–511, 1999.

[8] P. Flajolet and G.N. Martin. Probabilistic counting. In *IEEE Symposium on Foundation of Computer Science*, pages 76–82, 1983.

[9] Anna C. Gilbert, Sudipto Guha, Piotr Indyk, Yannis Kotidis, S. Muthukrishnan, and Martin Strauss. Fast, small-space algorithms for approximate histogram maintenance. In *ACM Symposium on Theory of Computing*, pages 389–398, 2002.

[10] Sudipto Guha, Nick Koudas, and Kyuseok Shim. Data-streams and histograms. In *ACM Symposium on Theory of Computing*, pages 471–475, 2001.

[11] Sudipto Guha, Nina Mishra, Rajeev Motwani, and Liadan O'Callaghan. Clustering data streams. In *IEEE Symposium on Foundations of Computer Science*, pages 359–366, 2000.

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

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

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

[15] Flip Korn, S. Muthukrishnan, and Divesh Srivastava. Reverse nearest neighbor aggregates over data streams. In *VLDB*, 2002.

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