Annotation and Computational Geometry in the Streaming Model*

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

May 5, 2003

Abstract

Computing over massive data streams has gained increasing importance in recent years. Set disjointness is a problem of particular interest because of its linear space requirement. To reduce the space requirement, we allow annotation to be added to the data stream. We show that the disjointness problem can be checked by a logarithmic-space verifier if linear-size annotation is added. We conjecture that linear-size annotation is necessary for any logarithmic-space disjointness verifier.

Besides stream annotation, we examine streaming space complexity for several problems in computational geometry. For one of these problems, the diameter problem, we introduced a small-space approximation scheme.

1 Introduction

With the fast development of computerized techniques for data collection, people and organizations are facing huge amounts of data. For example, sale records are collected everyday in

*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.

supermarkets. Phone companies keep track of calling records. Some network routers, such as the Cisco router with the NetFlow feature [Net98], are capable of producing network traffic reports continuously. While the router is routing the packets, the NetFlow feature can produce summary statistics for each network "flow."

However, it is not clear how to process and analyze this amount of raw data. People generally want to optimize their operations based on the information extracted from these massive data sets. The demand for applications that can perform massive-data processing tasks are increasing.

There are two commonly used algorithm-design paradigms for these problems, namely sampling and streaming [FKSV02a, FKSV02b]. A sampling algorithm takes a small random sample from the massive data set and computes some function from this sample. On the other hand, a streaming algorithm will take time to go through all the input. However, the streaming algorithm is only allowed little time to process each data element and little total workspace. It has to go through the data elements in a sequential order, and this order is not controlled by the algorithm.

Streaming is also a requirement in certain applications in which the data themselves have a stream nature. That is, the source generates data in the form of streams. It may not be necessary to place these data in persistent storage. The processing algorithm is required to be a streaming algorithm.

There has been significant progress recently in the design of streaming algorithms. In this report, we focus our attention on algorithm design and lower bounds for particular problems. For those high streaming space complexity problems, we examine approximation schema and possible annotations in order to lower the space requirement.

One of the problems of particular interest is the set disjointness problem. It is interesting not only because it has linear streaming space complexity but also because it is the problem from which people most often derive streaming space lower bounds for many other problems. We investigate the possibility of stream annotation for disjointness. Besides this, we also consider streaming algorithms and complexity for certain computational geometry problems.

2 Computational Model and Complexity Measure

An *data stream* is a sequence of data elements $\sigma_1, \sigma_2, \ldots, \sigma_n$ from a finite set M. Unless we explicitly say otherwise, we will denote by n the number of data elements in the stream and by m the maximum number of bits required to encode one data element.

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

1. It accesses the input data in a sequential order.
2. The order of the data elements in the stream is not controlled by the algorithm.

The complexity of a streaming algorithm is measured by the amount of workspace it requires and the time needed to process each data element. In some applications, the input data stream is not stored in persistent storage. The streaming algorithm that processes this data stream is required to go through the stream only once. In other applications, the algorithm may make multiple passes over the input. In the later case, we also measure the complexity of a streaming algorithm by the number of passes. Note that multiple-pass access is different from random access, because, in each pass, the algorithm is still required to access the input data in the given sequential order.

Because of the streaming nature of the input, it is important that little time be used to process each data element. Normally, the streaming model requires the processing time for one data element to be $O(\log n)$.

Usually, the stream model also requires sublinear space complexity. If linear space were allowed, the algorithm could keep all the input in memory, and there would be no need for the notion of "stream." On the other hand, there are many problems that require linear space. We will call this set of problems "hard" for the streaming model. In this scenario, approximation is necessary. Even for some sublinear-space problems, approximation may be used to further reduce the memory requirement. Unlike the situation in which approximation algorithms are used for **NP**-Complete problems, the purpose of approximation here is to trade accuracy for space.

3 Overview of Results

We investigate algorithms and lower bounds for some streaming problems. We also consider approximation and annotation. Here we summarize the results:

(1) We show that it is not possible to have sublinear-space streaming algorithm for set disjointness, even if multiple passes are allowed.

(2) Linear-size annotation can reduce the space requirement of a streaming checker for set disjointness to logarithmic-space.

(3) For geometric problems in the streaming model, we have the following bounds. Here, ϵ is a constant. Also, note that the $\Omega(n)$ bound doesn't mean that the space requirement is linear in the input size. Rather, the space requirement is linear in the number of points. The total input size is $n \cdot m$. In geometric problems, m is normally not independent of n but rather $O(\log n)$. The $\Omega(n)$ bound is thus for a total input size that is typically $O(n \log n)$.

Problem	one pass	multi-passes	one pass ϵ-approximation
Diameter	$\Omega(n)$	$\Omega(n)$	$O(\frac{m}{\epsilon})^*$
Closest Pair	$\Omega(n)$	$\Omega(n)$	$\Omega(n)$
K-Promised Convex Hull	$\Omega(n)$		

(* For inputs in two-dimensional Euclidian space.)

(4) We show that there is an algorithm that approximates the diameter of a set of points in k dimensional space within a factor of $1-\epsilon$ using $O(m(\frac{4\pi\sqrt{k-1}(1-\epsilon)}{\epsilon})^{k-1})$ space.

4 Related Work

The study of Turing machines with one-way read-only input tapes dates back to the 60's [HSL65, HU69]. These machines are close to the streaming model. In the years since, many algorithms for streaming computation have been devised. Most of these algorithms solve statistical problems. There are streaming algorithms for seletion [MP80, MRL98]. Flajolet and Martin [FM83] gave an algorithm for estimating the number of distinct elements in stream. Alon, Matias, and Szegedy [AMS99] showed how to approximate frequency moments. Feigenbaum, Kannan, Strauss, and Viswanathan [FKSV02a] showed how to approximate L_p differences. Indyk [Ind00] extended their work to compute L_p norms and differences for non-grouped data. There are also streaming algorithms for computing histograms [GKS01] and wavelet decompositions [GKMS01].

In database research, the concept of *synopsis data structures* is playing an increasingly important role. A synopsis [GM99] is a succinct approximate representation of the data in the DBMS. Queries can be answered based on the synopsis rather than on the (large) primary data set. Computing the synopsis and subsequently querying it are very similar to doing a streaming computation.

Apart from statistics and database queries, Guha, Mishra, Motwani, and O'Callaghan [GMMO00] showed how to cluster data streams, and Frieze and Kannan [FK99] showed how to approximate various matrix computations.

5 Preliminaries

In this section, we provide some of the technical background that we use in our results on stream annotation and geometric algorithms.

Let X, Y, and Z be arbitrary finite sets and $f : X \times Y \to Z$ be an arbitrary function. Alice and Bob are two players who want to evaluate the function f. However, Alice knows only $x \in X$ and Bob knows only $y \in Y$. They compute the function f using a communication protocol $\mathcal{P}$. Communication complexity studies the number of bits that Alice and Bob must exchange in order to evaluate a particular function f using a particular $\mathcal{P}$ [KN97].

Communication complexity is closely related to space complexity in the streaming model. There is a straightforward way to transform a streaming algorithm with space complexity $S(n)$ into a communication protocol with communication complexity $S(n)$. If a streaming algorithm exists for a problem, Alice can simulate the streaming algorithm on her input and

then transmit her memory contents to Bob. Bob can continue the streaming algorithm on his input. The amount of communication in this scheme is exactly the amount of memory used by the streaming algorithm. Thus communication complexity lower bounds for a particular problem are also space lower bounds for the same problem in the streaming model.

As we mentioned in section 1, "set disjointness" is the problem that has been used most often to derive lower bounds in streaming model.

Given a set U of size n and two subsets $x \subseteq U$ and $y \subseteq U$, we can define the following functions and problems.

Definition 5.1. ***Set Disjointness****: the function $disj(x, y)$ is defined to be "1" when $x \cap y = \phi$ and "0" otherwize. The corresponding language, DISJ, is the set $\{(x, y) | x \subseteq U,\ y \subseteq U,\ x \cap y = \phi\}$. In the streaming model, the input $(x,\ y)$ is given as a stream. Each of x and y can be represented by an n-bit string.*

Definition 5.2. ***Equality****: the function $eq(x, y)$ is defined to be "1" when $x = y$ and "0" otherwise. In the streaming model, the input $(x,\ y)$ is given as a stream.*

Definition 5.3. ***Index****: given an input stream consisting of a bit vector S of length n and a number $t \in [0,\ n-1]$ appended to the vector, output the bit S_t.*

The following communication-complexity lower bounds hold for these problems:

Theorem 5.1. [KS90] *The communication complexity of DISJ is $\Theta(n)$.*

Theorem 5.2. [KN97] *The one-round communication complexity of the index problem is $\Theta(n)$.*

Theorem 5.3. [KN97] *The deterministic communication complexity of the function eq is $\Theta(n)$, but the randomized communication complexity is $\Theta(\log n)$.*

From our discussion about the relationship between streaming-space complexity and communication complexity, the above bounds translate directly into one-pass streaming-space lower bounds for the corresponding streaming problems.

Our first result is that the linear-space lower bound holds for DISJ even if a constant number of passes is allowed.

Theorem 5.4. *DISJ requires linear space in the streaming model, even if $O(1)$ passes are allowed.*

Proof. The lower bound for one-pass stream algorithms can be derived by showing that, if we have a stream algorithm for DISJ that requires at most a certain amount of memory, then we are guaranteed to have a communication protocol requiring at most the same amount of communication.

Now, if we allow multiple passes, the above argument still works, except that the communication will be twice the amount of memory required by the stream algorithm multiplied by the

number of passes. Assume Alice and Bob possess inputs x and y respectively. In the middle of each pass, Alice sends her memory contents to Bob, and, at the end of each pass, Bob sends back his memory contents: two communications per pass. Thus the total communication will be twice of the amount of memory required by the stream algorithm multiplied by the number of passes.

Given the linear communication complexity of DISJ, it is not possible to have a sublinear-space stream algorithm that makes a constant number of passes. ■

6 Annotation of streams

We next ask whether streaming-space requirements can be reduced by adding some extra information to the input stream. Intuitively, the entity that generates the data stream may provide this extra information and help to reduce the resource requirements of the streaming algorithm. We model this situation with a stream proof system, in which we have both a streaming verifier and a prover. The prover has unlimited computing power and adds a proof (or annotation) to the stream. The verifier is a streaming algorithm that checks the proof.

The streaming verifier for a language L can be viewed as a probabilistic oracle machine M that obeys the streaming restriction and satisfies:

- *completeness*: For every stream $x \in L$, there exists a proof π_x:

$$Pr[M^{\pi_x}(x) = 1] \geq \frac{2}{3}$$

- *soundness*: For every stream $x \notin L$ and all proofs π

$$Pr[M^{\pi}(x) = 1] \leq \frac{1}{3}$$

We measure the complexity of this oracle machine both by the space requirement of the verifier and by the size of the proof.

Our next result is a stream proof system for DISJ.

Recall that we are trying to compute the function $disj(x, y)$ of two subsets $x \subseteq U$ and $y \subseteq U$ such that the function has value "1" when $x \cap y = \phi$ and "0" otherwise. Once x and y are chosen, each element i in the universe U will belong to one of the following 4 categories:

C1 $i \in x$ and $i \in y$;

C2 $i \in x$ and $i \notin y$;

C3 $i \notin x$ and $i \in y$;

C4 $i \notin x$ and $i \notin y$;

Because there are only 4 categories, a bit vector of size $O(|U|)$ can be constructed to show the categories of all of the elements in U.

Algorithm 6.1. Prover

The prover indicates, for each element in U, which of the above 4 categories it belongs to. This gives a bit vector p. It then appends p to the end of the input stream.

The input to the verifier is a stream of the form xyp. Because p indicates the category for each of the element in U, the verifier can reconstruct two subsets x' and y' from p. That is, $x' = \{i \in U | i \in \mathbf{C1}$ or $i \in \mathbf{C2}$ according to the proof $p\}$ and $y' = \{i \in U | i \in \mathbf{C1}$ or $i \in \mathbf{C3}$ according to the proof $p\}$. The verifier needs to check that $x = x'$ and $y = y'$. For this, it can use Lipton's set-fingerprinting technique [Lip90].

Given a multi-set $s = \{a_1, \ldots a_n\}$, where each a_i is an m-bits number, the fingerprint of this multi-set is computed by evaluating the following polynomial at some random location r.

$$F_s(x) = \prod_{i=1}^{n} (a_i + x)$$

The additions and multiplications are carried out in the field $\mathbb{F}_q$ where q is a prime selected randomly from the interval $[(nm)^2,\ 2(nm)^2]$.

Algorithm 6.2. Verifier

The stream verifier checks the following:

1. *There is no element of U in category **C1**. This can be checked trivially in one pass through p.*
2. *$x = x'$ and $y = y'$. This can be checked by randomly choosing r in $\mathbb{F}_q$ and checking that $F_x(r) = F_{x'}(r)$ and $F_y(r) = F_{y'}(r)$.*

The verifier accepts if and only if (1) and (2) both hold.

Theorem 6.1. [Lip90] *The probability that two sets $s_1 = \{a_1 \ldots a_k\}$ and $s_2 = \{a_1 \ldots a_l\}$ are unequal but have the same fingerprints is at most*

$$O(\frac{\log n + \log m}{nm} + \frac{1}{nm^2}),$$

where all elements in s_1 and s_2 are m-bit numbers, and $n = \max(k, l)$.

Note that for a set $s = \{a_1 \dots a_n\}$, the fingerprint can be computed in one pass, using $O(\log n)$ space.

Lemma 6.1. *The proof system given by Algorithms 6.1 and 6.2 satisfies the "completeness" and "soundness" requirements.*

Proof. It is clear that, if $x \cap y = \phi$, a truthful proof will make the verifier accept. On the other hand, if $x \cap y \neq \phi$, the prover can cheat with a proof p claiming that elements in category **C1** are in other categories. However, in this case, the resulting x' or y' will not be equal to the x or y in the original input. Such a discrepancy can be caught with high probability, according to **theorem 6.1**, by comparing fingerprints. ■

Theorem 6.2. *DISJ can be verified in $O(\log n)$ space if a linear-size annotation is provided.*

Proof. The proof has size $O(n)$, because it only needs to encode the category for each element in U. The only verification that requires working space is the check that $x' = x$ and $y' = y$. Because the fingerprint technique needs only $O(\log n)$ space [Lip90], the whole verifier needs only $O(\log n)$ space. ■

7 computational geometry in the streaming model

In many applications, the input data elements are interpreted as points in space. Geometric algorithms are used to reveal the relationships among the data elements. A typical example is clustering, in which similarity among data elements is discovered. In this section, we consider three geometric problems in the streaming model.

Given an input stream of points, we want to compute:

1. diameter
2. closest point
3. convex hull

As defined in section 2, an input stream is a sequence of data elements drawn from a finite set. In this section, the streams are sequences of points. We denote by n the number of points in the sequence and m the number of bits needed to encode each point.

Reductions are used in this section to derive lower bounds. As polynomial-time reductions are required for proving **NP**-completeness results, here we need reductions that can be computed in a streaming fashion. That is, the reduction itself should be a streaming algorithm. Given an input data stream, it produces another stream [BYKS02].

7.1 Diameter

Definition 7.1. *The **diameter** of a set of points is the maximum, over all pairs in the set, of the distance between the points in the pair.*

Theorem 7.1. *Any streaming algorithm that computes the diameter exactly requires $\Omega(n)$ space.*

Proof. We reduce the set-disjointness problem to a diameter problem. Consider points in two-dimensional space that are on a circle. 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 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 call (p_i, p_i') a pair and map each element $i \in U$ onto one such 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 subset 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 7.1

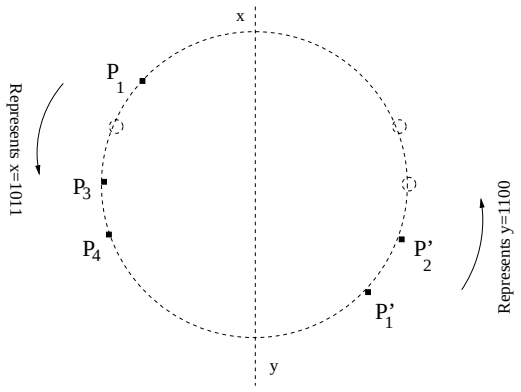

Figure 7.1: Reduction from DISJ to Diameter

The solid squares in the figure are the points we put into the diameter instance. The DISJ instance in Figure 7.1 is x, y, where $x = 1011$ and $y = 1100$. We map element i to the pair of points (p_i, p_i'). If the element i appears in x, the corresponding point p_i will be added to the stream. If it also appears in y, the point p_i' will be added to the stream too. 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 $\frac{1}{2n^2}$ must be detectable. This means that the encoding of each point has to have precision at least $\frac{1}{2n^2}$. Thus m must be $\Omega(\log n)$. For this reason, the $\Omega(n)$ bound doesn't mean that the space requirement is linear in the input size, because the input size is $\Omega(n \log n)$. However, it means that space linear in the number of points is necessary.

We now give an approximation algorithm for computing diameters in two-dimensional Euclidian space. A common approach toward approximation in streaming model is to divide the data set into groups. For each group, instead of keeping all the data in that group, we store only a summary of that data. The final result is then computed from the summaries of the groups. Our diameter-approximation algorithm takes this approach. We divide the space into sectors and compute the diameter for the point set using the information from each sector.

To construct the sectors, we locate a point x_0 and divide the plane into sectors centered at x_0. Each sector has an angle of θ. Two sectors are shown in figure 7.2

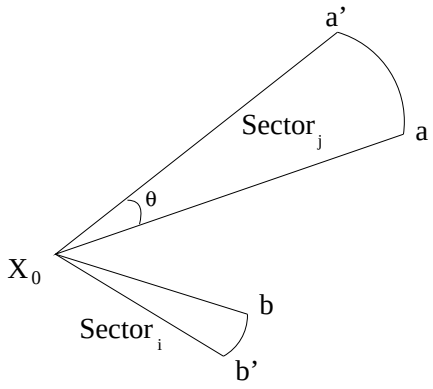

Figure 7.2: Two Example Sectors

The arcs aa' and bb' in the figure are the outer boundaries of the corresponding sectors. That is, all the points in the sector j are included in the area x_0aa'. Furthermore, there is at least one point on the boundary arc. Let u and v be two points in the space. Denote by $|uv|$ the distance between these two points. We also use the following notation:

x_0: the chosen center

R: the maximum distance between x_0 and any other point

θ: the angle we use to define the sectors

$D^{ij}_{max} = \max |uv|$ for $u \in$ boundary arc of sector i and $v \in$ boundary arc of sector j

$D^{ij}_{min} = \min |uv|$ for $u \in$ boundary arc of sector i and $v \in$ boundary arc of sector j

For each non-empty sector, the algorithm records the farthest point from the center in that sector. After scanning the input once, it estimates the diameter from these sectors.

Algorithm 7.1. 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, *between the center (the first point) and any other point.*
3. *Output* $\max\{R, \max_{i,j \in S} D^{ij}_{min}\}$ *as the diameter of the point set.*

Claim 7.1. *The distance between any two points in sector* i *and sector* j *is no larger than* $\max\{R, D^{ij}_{max}\}$. (*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 $|uv| \leq \max\{|x_0v|, |vu'|\} \leq \max\{R, |x_0v|, |u'v'|\} \leq \max\{R, D^{ij}_{max}\}$. ■

Claim 7.2. $D^{ij}_{max} \leq D^{ij}_{min} + length(aa') + length(bb') \leq D^{ij}_{min} + 2R \cdot \theta$

Proof. Let $|uv| = D^{ij}_{max}$ and $|u'v'| = D^{ij}_{min}$. Since $u, u' \in arc\ aa'$ and $v, v' \in arc\ bb'$, we observe that there is a path from u to v, namely $u \sim u' \sim v' \sim v$. Therefore $D^{ij}_{max} \leq |uu'| + D^{ij}_{min} + |vv'| \leq D^{ij}_{min} + 2R \cdot \theta$. ■

We use the same angle θ for all the sectors. Thus the only parameter in our algorithm that needs to be decided is θ. It is also related to the estimation error. We now calculate θ from the error requirement.

Assume that the true diameter *diam* is the distance between a point in sector i and another point in sector j. Let d be the diameter computed by our algorithm. Since the algorithm is allowed to make an error of ϵ, we need $(1 - \epsilon) \cdot diam \leq d \leq (1 + \epsilon) \cdot diam$.

On the other hand we have the following inequality:

$$\max\{R, D^{ij}_{min}\} \leq \max\{R, \max_{m,n \in S} D^{mn}_{min}\} = d \leq diam \leq \max\{R, D^{ij}_{max}\} \quad (1)$$

Depending on the relationship between R and D_{min}^{ij}, we have two cases:

Case 1: $R \geq D_{min}^{ij}$. In this case, inequality 1 becomes:

$$R \leq d \leq diam \leq D_{max}^{ij}$$

To bound the error, we require:

$$R \geq (1-\epsilon)D_{max}^{ij}$$

Take $R \geq (1-\epsilon)(R + 2R \cdot \theta) \geq (1-\epsilon)(D_{min}^{ij} + 2R \cdot \theta) \geq (1-\epsilon)D_{max}^{ij}$. This leads to $\epsilon R \geq (1-\epsilon)2R \cdot \theta$ and requires the angle θ of the sectors to be:

$$\theta \leq \frac{\epsilon}{2(1-\epsilon)}$$

Case 2: $R < D_{min}^{ij}$. In this case, inequality 1 becomes:

$$D_{min}^{ij} \leq d \leq diam \leq D_{max}^{ij}$$

Again, to bound the error, we require:

$$D_{min}^{ij} \geq (1-\epsilon)D_{max}^{ij}$$

Take $D_{min}^{ij} \geq (1-\epsilon)(D_{min}^{ij} + 2R \cdot \theta) \geq (1-\epsilon)D_{max}^{ij}$. This requires the angle θ of the sectors to be:

$$\theta \leq \frac{\epsilon}{2(1-\epsilon)} \leq \frac{\epsilon D_{min}^{ij}}{2(1-\epsilon)R}$$

Thus we define the sector size to be:

$$\theta \leq \frac{\epsilon}{2(1-\epsilon)} \tag{2}$$

Theorem 7.2. *There is an algorithm that approximates the diameter for a set of points in two-dimensional Euclidian space within* $1-\epsilon$ *and uses* $O(\frac{1}{\epsilon}m)$ *bits of space.*

Proof. Follows directly from our analysis of sector size. ■

In $\mathbb{R}^k$, Claim 7.1 still holds but Claim 7.2 becomes $D_{max}^{ij} \leq D_{min}^{ij} + 2\sqrt{k-1} \cdot R \cdot \theta$ and thus (2) changes accordingly to:

$$\theta \leq \frac{\epsilon}{2\sqrt{k-1}(1-\epsilon)} \tag{3}$$

The number of sectors in $\mathbb{R}^k$ increases to:

$$(\frac{4\pi\sqrt{k-1}(1-\epsilon)}{\epsilon})^{k-1} \tag{4}$$

In general, in k-dimensional Euclidian space, for any $0 < \epsilon < 1$, our algorithm can approximate the diameter with ϵ one side error using at most $m(\frac{4\pi\sqrt{k-1}(1-\epsilon)}{\epsilon})^{k-1}$ memory space. The algorithm suffers from the common problem of the "curse of dimensionality" in the sense that the space requirement increases exponentially with the dimension.

7.2 Closest Pair

Definition 7.2. ***Closest Pair*** *is the pair of points in the input stream the distance between which is the minimum among all pairwise distances in the stream.*

The closest-pair problem and the diameter problem are related. The former finds the minimum of the pairwise distances while the later finds the maximum. It is not surprising to see the following:

Theorem 7.3. *Any exact streaming algorithm for closest pair requires $\Omega(n)$ memory bits*

Proof. Again, we reduce from set disjointness. Consider points in 1-dimensional space. Construct a stream of this type of points from the disjointness instance. Given (x, y) as instance of disjointness, if the ith bit of x is "1," add to the stream the point with coordinate i. If it is "0," add nothing. For the subset y, if the ith bit of y is "1," the point $i - \epsilon$ is added, and, if it is "0," nothing is added.

If $x \cap y = \phi$, the minimal pairwise distance will be $1 - \epsilon$. On the other hand, if $x \cap y \neq \phi$, the minimal distance will be ϵ. By solving the closest-pair problem exactly, we could thus solve the set-disjointness problem. ■

Unlike the diameter problem, constant-factor approximation of the closest-pair problem is not easy. If there were an algorithm that approximated the minimum distance within some factor ϵ', one could always manipulate the ϵ in the stream construction described above. A proper ϵ could always be chosen to guarantee that approximation would allow us to distinguish the case of $x \cap y \neq \phi$ from the case of $x \cap y = \phi$.

7.3 K-Promised Convex Hull

The last problem we consider is convex hull.

Definition 7.3. *The* ***Convex Hull*** *of a set of points is the smallest convex set containing the points.*

Definition 7.4. *The* ***K-promised convex-hull*** *problem is a convex-hull problem in which the input point set is guaranteed to admit a convex hull with at most K sides.*

Once again, the space requirement for K-promised convex hull is $\Omega(n)$. The proof is a reduction from the index problem.

Theorem 7.4. *Streaming algorithms for K-promised convex hull require* $\Omega(n)$ *space.*

Proof. Given a bit vector S of length n, an algorithm that solves the index problem must return the bit S_i for some specified index i. Consider the points on two concentric circles, one of radius r and the other of radius $r - \epsilon$. For a point a_i on the inner circle, there is a point a_i' on the outer circle such that the center of the two circles lies on the line $a_i a_i'$. Call the pair (a_i, a_i') a "unit." We map the bits in S to the units. Namely, if $S_i = 1$, we put the point a_i' into the stream. Otherwise, a_i is added. We do this for all the bits in S. Note that different bits are mapped to different units that are distributed evenly along the circles. The result of this is a stream of points. At the end of the input, the index i is revealed, we add another $k - 1$ free points (*i.e.* points that can be place anywhere besides the circles) to the stream. The purpose of the $k - 1$ points is to build a convex set that includes the circles and uses exactly one point on one of the circles.

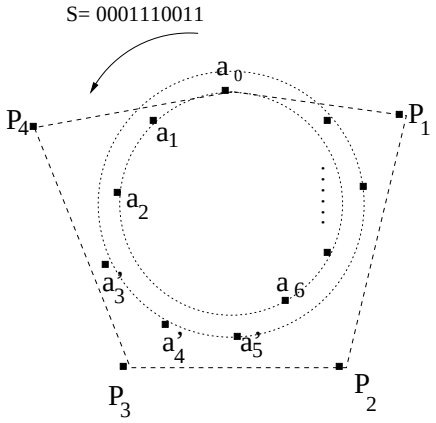

Figure 7.3: Convex Hull Using Only One Point on the Circles

Figure 7.3 gives us an example of such a construction. Again, the solid squares are the points we add to the stream. The example input bit vector is $S = 0001110011$. The index $i = 0$ and four free points p_1, p_2, p_3 and p_4 are placed such that the convex hull of the point set will be p_1, p_2, p_3, p_4 and a_0 or a_0'. In the example, the hull consists of $\{p_1, p_2, p_3, p_4, a_0\}$.

There will be only one point on the hull that is not in the set $\{p_1, p_2, p_3, p_4\}$. Once the convex hull is found, we can locate this point and deduce the bit value of S_i by calculating the distance from this point to the center of the circles.

Thus an algorithm that solved the k-promised convex-hull problem could be used to solve the index problem as well. ■

As in our construction for diameter, we need certain precision to ensure the computation of the convex hull and the detection of distance difference ϵ. This again requires m to be $O(\log n)$.

8 Open Problems

In section 6, we showed that disjointness can be checked with a logarithmic space verifier if linear-size annotation is allowed. One open problem in this direction is whether the size of the annotation can be improved. For example, could logarithmic-size annotation still enable a logarithmic-space verifier? We conjecture that, for logarithmic-space disjointness verifiers, linear-size annotation is necessary. On the other hand, annotation in general is complementary to approximation for small-space streaming algorithms. As shown by our preliminary results, it may also help to reduce the number of passes an algorithm needs. It would be interesting to see how annotation could be used in other problems.

In section 7, we provided an algorithm that approximates the diameter of a set of data in two-dimensional Euclidian space within a factor of $1-\epsilon$ using just $O(\frac{1}{\epsilon}m)$ memory space. However, the space requirement increases exponentially as the dimension grows. One open problem here is whether this increase is necessary. Would the same approximation scheme become applicable in high-dimensional space when dimension-reduction techniques are employed? A straightforward application of existing dimension-reduction methods does not seem to work well for this scenario. What would be a applicable dimension-reduction technique for this approximation scheme?

References

[AMS99] 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.

[BYKS02] Z. Bar-Yossef, R. Kumar, and D. Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *SODA*, pages 623–632, 2002.

[FK99] Alan M. Frieze and Ravi Kannan. Quick approximation to matrices and applications. *Combinatorica*, 19(2):175–220, 1999.

[FKSV02a] J. Feigenbaum, S. Kannan, M. Strauss, and M. Viswanathan. An approximate L^1 difference algorithm for massive data streams. *SIAM Journal on Computing*, 32:131–151, 2002.

[FKSV02b] J. Feigenbaum, S. Kannan, M. Strauss, and M. Viswanathan. Testing and spot-checking of data streams. *Algorithmica*, 34:67–80, 2002.

[FM83] P. Flajolet and G.N. Martin. Probabilistic counting. In *IEEE Symposium on Foundation of Computer Science*, pages 76–82, 1983.

[GKMS01] Anna C. Gilbert, Yannis Kotidis, S. Muthukrishnan, and Martin Strauss. Surfing wavelets on streams: One-pass summaries for approximate aggregate queries. In *The VLDB Journal*, pages 79–88, 2001.

[GKS01] Sudipto Guha, Nick Koudas, and Kyuseok Shim. Data-streams and histograms. In *ACM Symposium on Theory of Computing*, pages 471–475, 2001.

[GM99] P. Gibbons and Y. Matias. Synopsis data structures for massive data sets. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science: Special Issue on External emory Algorithms and Visualization*, A:39–70, 1999.

[GMMO00] 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.

[HSL65] J. Hartmanis, R. Stearns, and P.M. Lewis. Hierarchies of memory limited computations. In *IEEE Conf. Record on Switching Circuit Theory and Logica Design*, pages 179–190, 1965.

[HU69] J.E. Hopcroft and J.D. Ullman. Some results on tape-bounded turing machines. *Journal of the ACM*, 16:160–177, 1969.

[Ind00] P. Indyk. Stable distributions, pseudorandom generators, embeddings and data stream computation. In *IEEE Symposium on Foundations of Computer Science*, pages 189–197, 2000.

[KN97] E. Kushilevitz and N. Nisan. *Communication Complexity*. Combridge University Press, 1997.

[KS90] B. Kalyanasundaram and G. Schnitger. The probabilistic communication complexity of set intersection. *SIAM Journal on Discrete Math.*, 5:545–557, 1990.

[Lip90] R.J. Lipton. Efficient checking of computations. In *STACS*, pages 207–215, 1990.

[MP80] J.I. Munro and M.S. Paterson. Selection and sorting with limited storage. *Theoretical Computer Science*, 12:315–323, 1980.

[MRL98] Gurmeet Singh Manku, Sridhar Rajagopalan, and Bruce G. Lindsay. Approximate medians and other quantiles in one pass and with limited memory. In *SIGMOD*, pages 426–435, 1998.

[Net98] Cisco NetFlow. http://www.cisco.com/warp/public/732/netflow/, 1998.