Streaming and Dynamic Algorithms for Minimum Enclosing Balls in High Dimensions

by

Vinayak Pathak

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2011

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

At SODA'10, Agarwal and Sharathkumar presented a streaming algorithm for approximating the minimum enclosing ball of a set of points in d-dimensional Euclidean space. Their algorithm requires one pass, uses $O(d)$ space, and was shown to have approximation factor at most $(1+\sqrt{3})/2+ \approx 1.3661$. We prove that the same algorithm has approximation factor less than 1.22, which brings us much closer to a $(1+\sqrt{2})/2 \approx 1.207$ lower bound given by Agarwal and Sharathkumar.

We also apply this technique to the dynamic version of the minimum enclosing ball problem (in the non-streaming setting). We give an $O(dn)$-space data structure that can maintain a 1.22-approximate minimum enclosing ball in $O(d \log n)$ expected amortized time per insertion/deletion.

Finally, we prove that a $1+\varepsilon$ approximation to the problem can be found in $\frac{0.5+\delta}{\varepsilon}$ passes over the input, for an arbitrarily small constant δ, which is an improvement over the previous result that used $\lfloor 2/\varepsilon \rfloor$ passes.

Acknowledgements

I thank Prof Timothy Chan for being an excellent supervisor. It has been a great learning experience.

Table of Contents

List of Figures

Chapter 1

Outline

In this thesis, we study the problem of finding the ball of minimum radius that encloses a given set of points in high dimensions. Our results are based around two different but not unrelated themes—streaming algorithms and dynamic algorithms.

Streaming algorithms [22, 5] are algorithms where the input is provided in a stream. The algorithm is allowed to examine it a small number of times (called passes) and do some computation on it. In addition, the space available is restricted to be sublinear. This is natural because with linear space available, the algorithm would just store all of the input in its memory in one pass and thus the streaming model of computation would be no different from the usual model. The performance of a streaming algorithm is usually measured in terms of the space used. Also, since many problems are not solvable exactly in this setting, another measure of performance is the approximation ratio.

Dynamic algorithms [14] are algorithms where the input is dynamic, i.e., objects can be added to the input or deleted from it at any time. The task is to maintain a data structure that supports insertions, deletions and some kind of queries over the input currently available. Here, the space available is not as restrictive as in the case of streaming algorithms. The measures for the performance of a dynamic algorithm are the update and query times achieved. Once again, if the problem at hand is not solvable exactly, then the

approximation ratio is also an important measure of performance.

For the case of streaming algorithms, we investigate the approximation ratio achieved by an algorithm by Agarwal and Sharathkumar [4] that appeared in SODA'10. Their paper claimed an approximation factor of at most $\frac{1+\sqrt{3}}{2} < 1.3661$. We prove that the same algorithm achieves a much better factor of 1.22, which is much closer to the lower bound of $\frac{1+\sqrt{2}}{2} > 1.207$, proved by Agarwal and Sharathkumar. The proof for the factor 1.22 is computer-assisted, i.e., we wrote a program in C to do some computation on a large number of carefully chosen values of carefully chosen parameters. A proof of factor < 1.2308 is also provided that does not need any computer assistance.

For the case of dynamic algorithms, we combine ideas from Agarwal and Sharathkumar's paper with some previously known techniques to prove that the same approximation ratio of ≈ 1.22 can be achieved in the dynamic setting too. No upper bound was known in the literature before our result; however, a trivial factor-2 algorithm was easily achievable.

The main content of the thesis is derived from the paper I wrote with my supervisor Timothy Chan, titled "Streaming and Dynamic Algorithms for Minimum Enclosing Balls in High Dimensions," which is to appear in *Proc. 12th Algorithms and Data Structures Symposium (WADS),* 2011.

The thesis is organized as follows. In Chapter 2, we introduce the streaming model by giving some motivation for why it is studied and then discuss a few examples of problems from the literature. We also discuss the geometric problems that have been studied in the streaming model and briefly discuss the techniques that are relevant for understanding our results. In Chapter 3, we describe our first result, i.e., the better analysis of Agarwal and Sharathkumar's algorithm and in Chapter 4, we describe our result in the dynamic settings. Finally, Chapter 5 mentions additional minor results and observations and also discusses some future research directions. In the Appendix, we provide the C code that was used for achieving factor 1.22.

Chapter 2

Streaming algorithms

Consider the task of earthquake prediction. According to the Wikipedia article [26],

> Many phenomena are considered to be possible precursors of earthquakes, and among those under investigation are seismicity, changes in the ionosphere, various types of electromagnetic indicators including infrared and radio waves, radon emissions, and even unusual animal behavior.

For a phenomenon that is affected by so many factors, any serious prediction system should probably have thousands of sensors placed all over the earth, under the surface of the earth and even in outer space, in satellites revolving around the earth. These sensors will generate massive amounts of data per second and will send them to our earthquake prediction center. Our task will be to somehow process the plethora of data and detect useful patterns in it so that we can predict the occurence of earthquakes with a sufficient accuracy. In this situation where the stakes are so high, all responsibility will rest on the shoulders of the system we have implemented at the prediction center and the computation our computers are going to do on the data that arrives.

Because of the technological development in the last few centuries, such complexity has become ubiquitous and that has forced us to look for smart ways to put a handle on the

data all around us. One way, of course, is to design faster and more efficient processors. The other one is just to develop better algorithms for solving these problems. In fact, there is evidence that the second method might be better. According to a report [1] written in order to decide the US government's policy towards funding scientific research, it was observed that progress in algorithms has beaten Moore's law. To quote from the report:

> ... a benchmark production planning model solved using linear programming would have taken 82 years to solve in 1988, using the computers and the linear programming algorithms of the day. Fifteen years later—in 2003—this same model could be solved in roughly 1 minute, an improvement by a factor of roughly 43 million. Of this, a factor of roughly 1,000 was due to increased processor speed, whereas a factor of roughly 43,000 was due to improvements in algorithms!

A streaming algorithm is a special kind of algorithm where we assume that the whole input, just like in our example of the earthquake prediction, is being given in the form of a stream. That is, we are not allowed random access to its bits and we have only $o(n)$ memory available with us, thus we cannot store the whole input in the memory. Researchers in this area try to understand the kinds of problems that can be solved in such a restrictive model, and if they cannot be solved exactly, the least amount of compromise in accuracy that can be achieved. In addition to their practical usefulness, streaming algorithms have led to the development of some very nice mathematical tools.

2.1 Examples

In this section, we discuss some simple examples of problems that have been studied in the streaming model in order to get a rough idea of what can and cannot be achieved with streaming algorithms. A more comprehensive discussion on streaming can, for instance, be found in Muthukrishnan's survey [22].

2.1.1 Counting

Given an unknown number of items, count the number of them.

Here, the space is measured in terms of the number of items, i.e., if the number of items turns out to be n, then at no point should we have used space $\Omega(n)$. Obviously, with logarithmic space, we can maintain the count. Thus this problem can be easily solved in the streaming model. In fact, if we allow some probabilistic error, it can be solved [16] with a good enough accuracy in $O(\log \log n)$ space.

2.1.2 Missing number

Given n distinct integers in a stream from the set $\{1, \ldots, n+1\}$, find out which one is the missing number.

Obviously, with linear space, we can store all the numbers, sort them and then do a linear scan from left to right to find out the missing one. But with sublinear space, the trick [22] is to add the numbers we have seen till now and store their sum in a register. Since the maximum sum possible is $(n+1)(n+2)/2$, it needs at most a logarithmic size register to be stored. In the end, we just need to subtract the sum we have from the sum of all numbers from 1 to $n+1$. This will give the missing number.

2.1.3 Number of distinct items

Given m items, each drawn from the universe $U = \{1, \ldots, n+1\}$, find the number of distinct items.

This problem is trickier. At any given moment, when a new item arrives in the stream, we need some way to decide whether we had already seen this item or not. The item could have occured long ago or very recently. To retrieve this information, we may need to store all the items we have seen till now, which is not allowed.

It is not difficult to see that this problem cannot be solved exactly and deterministically using $o(n)$ space. Suppose that we have solved a particular instance with $m-1$ items. We claim that an adversary can find out the subset of U we have received from the information we have stored in our memory. To find out if we have received item i, the adversary can give our algorithm item i as the next input of the stream and ask for the output again. Item i is in the subset before this step if and only if the number of distinct items does not increase. If we have enough information to identify the subset of U exactly, then we must be using linear space. (This can be made more formal using a counting argument.)

If we are allowed to make some error with a low probability, however, it is possible to solve this problem in the streaming model [17]. The intuition is as follows. Suppose you have gone fishing to a nearby lake and there are roughly 10,000 fish in the lake belonging to five different species. You also know that some of the species are rare, say, for example, there are only five salmons in the whole lake. Now if each fish is equally likely to get caught in your bait, catching a salmon is an indication that you have caught a large number of fish.

Thus the trick is to divide the universe U into different disjoint subsets $S_0, \ldots, S_k$, where k is $o(n)$ such that the size of set S_i decreases as we increase i. A subset corresponds to a species from the example. Thus as i increases, the species becomes more rare. If we can determine the set in which an item $j \in U$ lies efficiently, we can use this information to solve the problem. We just maintain the largest value of i such that we have seen an item in the stream from the subset S_i. The larger the value, the more number of items we must have seen. One convenient subdivision of U we can use is the following. Let S_i contain all numbers from the set U that end in exactly i zeroes (when written in, say binary). This would work, except that we need the items from U to come with a uniform probability distribution over U. However, the adversary can easily fool us by giving many items from S_k in the beginning of the stream. The solution is to pick a random permutation of the n items and apply that to the input before feeding it to the algorithm. A random permutation can be applied in the streaming model using techniques from the hashing literature.

2.1.4 General frequency moments

The number of items and the number of distinct items are specific cases of a more general property of any sequence of items drawn from a given universe. These properties are called frequency moments.

Associate with each member x_i of the universe U, a frequency value f_i denoting the number of times you have seen that member. Then, the k^{th} frequency moment of the input stream, denoted F_k, is defined as $\sum_{i=1}^{n} f_i^k$. We can see that the number of items is F_1 and the number of distinct items is F_0. Alon et al. [6] studied these problems in their Gödel prize winning paper and gave a streaming algorithm for finding F_2 as well. They also proved that for $k \geq 6$, it's not possible to find F_k. Later, Indyk et al. [20] gave streaming algorithms for $k = 3, 4$ and 5 as well.

2.2 Streaming algorithms for geometric problems

Many problems on sets of numbers can be generalized to geometric problems about points point in d-dimensional space. For example, one way to generalize one-dimensional predecessor search is nearest neighbor search. Thus geometric problems arise quite naturally and a lot of them have been studied in the streaming model. Examples include minimum enclosing balls, minimum enclosing cylinders, minimum volume bounding box, minimum spanning tree, minimum weight matching and orthogonal range searching [19, 7].

In the next few sections, we describe some general techniques and approaches used in the literature for solving geometric problems in the streaming model. The discussion will be helpful in understanding our result in Chapter 4.

2.2.1 Core-sets

Before being used in streaming, *core-sets* were a popular tool for developing approximation algorithms for a wide range of geometric problems. Informally, a core-set of a given set P

of points with respect to an optimization problem is a subset of P of small size such that the solution to the problem on the subset gives a good approximation to the solution on P. If in addition, the core-set can be found with an efficient algorithm, it gives rise to a good approximation algorithm. Since it has a small size, one can use an inefficient algorithm for solving the optimization problem exactly on the core-set. By the definition of core-set, this will be a good approximation to the solution on the original set.

ε-Approximations

For many counting problems, an *ε-net* or an *ε-approximation* can play a role similar to a core-set, i.e., the solution of the problem on, say, the ε-approximation of the set can be a good approximation to the solution on the original set. An example is *orthogonal range searching*. Given a set P of n points in $\mathbb{R}^2$, we want to store them in a data structure in such a way that given an axis-parallel rectangle, one can report the number of points lying inside it with a good accuracy. There are many exact algorithms available for this problem, but they all require at least linear space [15, 23]. However, using an ε-approximation, one can solve this problem within an additive error using space depending only on the amount of error.

ε-Approximations are a general concept introduced first by Vapnik and Chervonenkis [24]. Consider a set S, and a set $R \subseteq 2^S$ of some of its subsets. The tuple $X = (S, R)$ is called a *range space*. Given $P \subseteq S$, an ε-approximation of P for the range space X is a set $Q \subset P$, such that for any range $r \in R$,

$$\left| \frac{|Q \cap r|}{|Q|} - \frac{|P \cap r|}{|P|} \right| \leq \varepsilon.$$

Consider the range space where S consists of all points in the two-dimensional plane and R consists of all subsets r such that there exists an axis-aligned rectangle that contains only the points in r and nothing else. Given any n point subset P of S, if we could find an ε-approximation Q for P with respect to the above range space, then the solution to the

range counting problem on Q would be a good approximation to the solution to the range counting problem on P. In particular, for any rectangle r,

$$\left|\frac{|P|}{|Q|}|Q \cap r| - |P \cap r|\right| \leq \varepsilon n.$$

By a standard theorem [24], for any range space with bounded *VC-dimension*, any random sample of P of size $O(\frac{1}{\varepsilon^2}\log\frac{1}{\varepsilon})$ is an ε-approximation with high probability. It is well known that the range space formed by axis-aligned rectangles has a bounded VC-dimension. Thus we can find an ε-approximation Q of P by just picking a random sample of P.

This technique is useful for the streaming model as well because sampling in the streaming model can be easily done by an online sampling method, such as the reservoir sampling method of Vitter [25]. Thus ε-approximations give us a constant-space algorithm for solving this problem up to an additive error.

Unfortunately, ε-approximations are not useful for many non-counting problems such as finding the width[1] of a given set of points. Moreover, the approximation achieved in the case of orthogonal range counting is also of a weak, additive kind.

To solve problems such as width, Agarwal et al. [3] came up with a general framework called the ε-kernels.

ε-Kernels

Let $\mathbb{S}^{d-1}$ denote the unit sphere centered at the origin in $\mathbb{R}^d$. For a given set P of points, we define the directional width of P in a direction $u \in \mathbb{S}^{d-1}$ as

$$w(u, P) = \max_{p \in P}\langle u, p\rangle - \min_{p \in P}\langle u, P\rangle,$$

[1] We define width in the next section.

where $\langle \cdot, \cdot \rangle$ is the standard inner product. For a given parameter ε, a subset $Q \subseteq P$ is called an *ε-kernel* of P if for all $u \in \mathbb{S}^{d-1}$,

$$\max_{p \in P} \langle u, p \rangle - \max_{q \in Q} \langle u, q \rangle \leq \varepsilon \cdot w(u, P). \tag{2.1}$$

If such a set Q exists and is of small size, then it will be useful for many problems. Consider, for example, the problem of finding the width of a given set of points. The width of a set P is defined as

$$\text{width}(P) = \min_{u \in \mathbb{S}^{d-1}} w(u, P).$$

Clearly, $\text{width}(P) \geq \text{width}(Q) \geq (1 - \varepsilon)\text{width}(P)$. Thus $(1 + \varepsilon)\text{width}(Q)$ is a $(1 + \varepsilon)$-factor approximation to $\text{width}(P)$. Agarwal et al. [3, 2] proved that an ε-kernel gives a $(1 + O(\varepsilon))$-factor approximation to numerous problems, including width, diameter, radius of the minimum enclosing ball and the volume of the minimum bounding box. They also gave an efficient algorithm for actually computing an ε-kernel of a given set of points.

The idea is simple. Assume without loss of generality that all points lie in the region $[-1, 1]^d$. Build a grid with each cell having a side length ε and pick one point of P arbitrarily from each non-empty grid cell. Let this set be the ε-kernel Q. Since the total number of grid cells is $O(1/\varepsilon^d)$, the size of the resulting set will be bounded by $O(1/\varepsilon^d)$. Of course, Q approximates P in some sense. In particular, in any direction u, $w(u, Q) \geq w(u, P) - \varepsilon\sqrt{d}$. However, this does not give us an ε-kernel according to (2.1) because the error here is additive as opposed to multiplicative. However, if we can ensure $w(u, P) \geq c$ for all directions u, where c is a constant, then we can get a multiplicative error of $1 + \varepsilon$ for a re-adjusted ε. This is guaranteed if P satisfies the conditions for being *α-fat* for some constant α. A point set P is called α-fat if there exists a point $p \in \mathbb{R}^d$ and a hypercube $\bar{\mathbb{C}}$ centered at the origin so that $p + \alpha\bar{\mathbb{C}} \subset \text{conv}(P) \subset p + \bar{\mathbb{C}}$ where $\text{conv}(P)$ denotes the convex hull of P.

But what if the given point set is not α-fat to begin with? Agarwal et al. [2] showed that any point set P can be converted into a point set P' that is α-fat by performing an affine transformation f such that $Q \subset P$ is an ε-kernel of P if and only if $Q' = f(Q)$ is

an ε-kernel of $P' = f(P)$. The transformation is computed using a known approximation algorithm for minium bounding box by Baraquet and Har-Peled [8] (See Section 4.1 for more details). Thus to find an ε-kernel of P, we first make P α-fat and then use the grid approach described above.

This gives us an ε-kernel of size $O(1/\varepsilon^d)$. This can immediately be improved by making the simple observation that the width along any direction is determined by the extreme points in that direction. Thus we do not need to store one representative point from all non-empty grid cells. We can just pick one representative point from the top-most non-empty cell and one from the bottom-most non-empty cell for each column. This modification gives us an ε-kernel of size $O(1/\varepsilon^{d-1})$. This bound was later improved by Chan [11] and independently by Yu et al. [27] to $O(1/\varepsilon^{(d-1)/2})$.

2.2.2 Merge-and-reduce

The algorithm described in the previous section for finding the ε-kernel of a set of points suffers with one disadvantage—it does not work in the streaming model. Also, although we had a streaming algorithm Section 2.2.1 for finding an ε-approximation of a set of points, it was randomized. There do exist deterministic algorithms for finding the ε-approximation but they do not work in streaming.

Both these issues can be fixed by applying the general technique of *merge-and-reduce*, which has its origins in the work by Bentley and Saxe [9] (the "logarithmic method") on dynamic data structures[2]. The technique works specifically for problems where we have some way of finding a "sketch" of small size of the input that approximates the original input in a way that is relevant for the problem at hand. As long as the sketch satisfies certain decomposability conditions, we can get a streaming algorithm for the same problem by calling the sketch finding algorithm as a black box. In the following paragraphs, we will demonstrate the method for the case of ε-kernels while keeping in mind that it is much

[2]As we will see later, dynamic data structures have much in common with the streaming model.

more general and applies to many other cases.

It is not difficult to verify that ε-kernels satisfy the following two properties:

1. If P_2 is an ε-kernel of P_1 and P_3 is an δ-kernel of P_2, then P_3 is a $(\delta + \epsilon)$-kernel of P_1;
2. If Q_1 is an ε-kernel of P_1, and Q_2 is an ε-kernel of P_2, then $Q_1 \cup Q_2$ is an ε-kernel of $P_1 \cup P_2$.

Since we are dealing with the streaming model, suppose that we have only $O(\sqrt{n})$ space. The way merge-and-reduce works is as follows.

Keep storing the stream in the memory until it is full. Once it is full, replace the $O(\sqrt{n})$ points in it with their ε-kernel. The ε-kernel is of constant size and thus the memory is almost empty now. Next, do the same with the new points. Let $P_1, P_2, \ldots, P_k$ be the groups of $O(\sqrt{n})$ points each. There are in total $O(\sqrt{n})$ of these groups in the input. At any point in the algorithm the memory will contain the following two things—the ε-kernels of all the groups already seen, and the points received so far in the next group of $O(\sqrt{n})$ points. Thus the memory will be full of ε-kernels once k reaches $O(\sqrt{n})$. But fortunately, that will happen only when the whole stream has arrived. At this point, we just find the union of the ε-kernel of all the P_i's. Because of the decomposability properties mentioned above, the result will be an ε-kernel for the original set of points for some re-adjusted ε.

We can reduce the memory requirement even further by increasing the number of "levels" in our algorithm. The essential idea is to call the ε-kernel finding subroutine more frequently. This will make us run out of memory before the whole stream is seen. When that happens, we replace the ε-kernels stored in the memory with the ε-kernel of the union of ε-kernels and continue. The best bound achievable this way on the size of memory is polylogarithmic in n. Details can be found in [2].

However, with merge-and-reduce, we can reduce the space to only polylogarithmic in terms of the input size. To improve the space bound even further, we need other techniques. In particular, Chan [11] gave an algorithm for maintaining the ε-kernel in the streaming

model using only a constant amount of space. His techniques are closely related to the paradigm of "doubling", which we describe in the next section.

2.2.3 Doubling

The paradigm of doubling exploits the fact that the sum of the terms of a geometric series is of the same order as the largest term. In this section, we demonstrate an application of this by describing an algorithm for maintaining the minimum enclosing cylinder of a set of points in $\mathbb{R}^d$ in the streaming model.

Approximate versions of this problem can be solved in low dimensions using ε-kernels. Because of the existence of streaming algorithms for maintaining the kernel, the problem can be approximately solved even in streaming as long as the dimension is low. However, because of the exponential dependence of the size of the ε-kernel on d, these algorithms are not suitable for high dimensions.

We first show how to get a constant factor approximation algorithm for the minimum enclosing cylinder problem in high dimensions. Let $o \in P$ be an arbitrary point in P and let $v \in P$ be the point farthest away from o. Assume that ov is the axis of the enclosing cylinder and return the distance of the farthest point from ov as the radius. It can be proved that this gives a factor-4 algorithm. In fact, something more general can be proved. If for all points $p \in P$, the ratio $\|op\|/\|ov\|$ is bounded by c from above, then we get an approximation algorithm with a factor of $2(c+1)$. The reason is the following observation taken from [11], which states that for any three points o, p and v,

$$d(p, ov) \leq 2\left(\frac{\|op\|}{\|ov\|} + 1\right) \mathrm{rad}(\{o, v, p\}). \tag{2.2}$$

Here, $d(p, ov)$ is the perpendicular distance between the point p and the line ov and $\mathrm{rad}(P)$ denotes the radius of the minimum enclosing cylinder of the set P. Since $\mathrm{rad}(\{o, v, p\})$ is a lower bound for $\mathrm{rad}(P)$, $\max_{p \in P} d(p, ov)$ is a good approximation to the radius of the minimum enclosing cylinder.

Unfortunately, this algorithm requires two passes—first pass to pick o and v, and the second pass to find $\max_{p \in P} d(p, ov)$. Doubling helps in doing both these things in just one pass.

First, notice that instead of maintaining $d(p, ov)$, one can maintain the maximum value of w_f of $\text{rad}(\{o, p, v\})$ in the second pass and in the end, report the radius as $2\left(\frac{\|op\|}{\|ov\|} + 1\right) w_f$. To combine the two passes in one, the basic idea is the following. Let $p_1, p_2, \ldots, p_n$ be the stream. Assume in the beginning that p_1p_2 is an approximately correct axis, i.e., $o = p_1$, $v = p_2$ and for any point $p \in P$, $\|op\|/\|ov\| < 2$. Next, maintain the maximum value of $\text{rad}(\{o, p, v\})$ until it stops being the correct axis, i.e., you get a point p_i so that $\|op_i\|/\|ov\| \geq 2$. In that case, change the axis to p_1p_i, i.e., set $v = p_i$ and continue.

If we never change v, the initial choice of the axis was already correct. If we do change v, then let $v_1, v_2, \ldots, v_f$ be the values that v attains over the course of the algorithm. Observe that $\|ov_i\| \geq 2\|ov_{i-1}\|$ for each i. For points that came after v_f, we know that $d(p, ov_f) \leq 2\left(\frac{\|op\|}{\|ov_f\|} + 1\right)\text{rad}(\{o, p, v_f\}) \leq 6w_f$. For points that came between v_f and v_{f-1}, $d(p, ov_f)$ could be large because during that time, ov_{f-1} was the assumed axis and not ov_f. However, it can be proved that it couldn't have been too large when compared to w_f. The reason is as follows. Let $\hat{p}$ be the projection of p on ov_{f-1}. Then, from triangle inequality,

$$d(p, ov_f) \leq d(p, ov_{f-1}) + d(\hat{p}, ov_f).$$

Also, since $\|ov_{f-1}\|/\|ov_f\| \leq 1/2$,

$$d(v_{f-1}, ov_f) \leq 2 \cdot 3/2 \cdot \text{rad}(\{o, v_{f-1}, v_f\}) \leq 3w_f.$$

Using similarity of triangles, we get that $\|o\hat{p}\|/\|ov_{f-1}\| = d(\hat{p}, ov_f)/d(\hat{v_{f-1}}, ov_f)$. Combining all these, we get

$$d(p, ov_f) \leq d(p, ov_{f-1}) + \frac{\|op\|}{\|ov_{f-1}\|} 3w_f.$$

We know a bound for $d(p, ov_{f-1})$ because when we received p, the assumed axis was ov_{f-1}. In general, for each p that came between v_1 and v_2, $d(p, ov_f)$ will be different from $d(p, ov_1)$, but not very different, the reason being that the error accumulated in step i is inversely

proportional to $\|ov_i\|$ and thus forms a geometric progression. Finally, after working out the details, it can be shown that this gives an approximation algorithm with factor 18 for the minimum enclosing cylinder problem.

Chapter 3

Minimum enclosing ball in streaming

3.1 Introduction

We now turn our focus to the main problem of the thesis—finding the minimum enclosing ball of a given set of points.

Let P be a set of points in $\mathbb{R}^d$. We use MEB(P) to denote the minimum enclosing ball of the set P, i.e., the ball with the smallest radius that encloses all points in P. For a ball B, we use $r(B)$ and $c(B)$ to denote its radius and center respectively. Let αB denote the ball with center at $c(B)$ and radius equal to $\alpha r(B)$.

A very simple factor-2 streaming algorithm for approximating the MEB works as follows. Let the first point be p_0. Find the point p_1 in P that is farthest away from p_0. This can be implemented by a one-pass streaming algorithm. Return the ball centered at p_0 of radius $\|p_0p_1\|$. This ball clearly encloses P. The approximation factor is at most 2, since the MEB of P must enclose p_0 and p_1, and any ball that encloses p and q must have radius at least $\|p_0p_1\|/2$.

One approach that leads to a factor of 1.5 is due to Chan and Zarabi-Zadeh [28]. The algorithm maintains a ball and updates it when it sees a point that does not lie inside it.

The initial ball is the MEB of the first two points. Whenever a point arrives that does not lie in the current ball, it replaces it with the MEB of the current ball and the new point.

In low dimensions, we can simply build an ε-kernel for the set of points in one pass to obtain a $(1+\varepsilon)$-factor approximation for finding the minimum enclosing ball for a re-adjusted ε. However, when dimension is large, the ε-kernel technique is not applicable since the size of the kernel increases exponentially with d. So what is the best we can do in high dimensions? Can we get an approximation factor better than 1.5?

One idea to explore is the possibility of having coresets whose size does not increase exponentially with d.

3.2 Core-set in high dimensions

A core-set, as described in Section 2.2.1, is a small subset of a given set of points such that the solution to the problem at hand on the core-set approximates the solution to the problem on the original set. Let us make the notion formal for the case of minimum enclosing balls.

Definition 1. *Given a set P of points in $\mathbb{R}^d$, an* ε-core-set *of P is a subset $Q \subseteq P$ such that $P \subseteq (1+\varepsilon)\mathrm{MEB}(Q)$.*

Bădoiu and Clarkson [10] proved that it is indeed possible to get such a core-set for minimum enclosing balls in high dimensions. In particular, given a set P of n points in $\mathbb{R}^d$ and a constant $\varepsilon > 0$, there exists a subset $Q \subset P$ such that $|Q|$ depends only on ε and $P \subset (1+\varepsilon)\mathrm{MEB}(Q)$. Moreover, Q can be found efficiently.

The algorithm is iterative. We start with just the set Q_0 containing one arbitrary point $p_0 \in P$. To update the core-set Q_i after i iterations, find the point in P that is farthest away from $c(\mathrm{MEB}(Q_i))$ and add that to Q_i. Bădoiu and Clarkson proved that this gives a

$(1+\varepsilon)$-core-set in $O(1/\varepsilon)$ iterations. Since each iteration adds one extra point to the core-set, the size of the final core-set must be $O(1/\varepsilon)$. The running time for the above algorithm is clearly polynomial in n. However, this algorithm does not work in the streaming model.

3.3 Coreset in high dimensions in streaming?

We saw earlier that a core-set finding algorithm can be converted to a core-set finding algorithm in streaming as long as the core-set satisfies certain decomposability properties. Unfortunately, the core-set from Definition 1 does not satisfy those properties. In particular, if Q_1 is a core-set of P_1 and Q_2 is a core-set of P_2, then it is not necessarily true that $Q_1 \cup Q_2$ is a core-set of $P_1 \cup P_2$. The following paragraph describes a counter-example.

Let P_1 be many points along the circumference of a circle centered at the origin in 2-d and P_2 be one point far away on the x-axis. Assume that ε is very small and thus the MEB of the core-set is almost the same as the MEB of the whole set. Then a pair of diametrically opposite points in P_1 such that the line joining them is close to vertical qualifies to be a core-set of P_1. Also, a core-set of P_2 is the point constituting P_2 itself. However, the MEB of the union of these two core-sets does not cover all points in P_1 if P_2 is far enough.

This shows that we cannot directly apply merge-and-reduce to the core-sets in high dimensions.

3.4 Agarwal and Sharathkumar's algorithm

At SODA'10, Agarwal and Sharathkumar [4] presented a streaming algorithm for approximating the minimum enclosing ball of a set of points in d-dimensional Euclidean space. Their algorithm requires one pass, uses $O(d)$ space, and was shown to have approximation factor at most $(1+\sqrt{3})/2 < 1.3661$, which was better than the trivial factor-2 algorithm that we mentioned in Section 3.1.

Their algorithm borrows ideas from both the merge-and-reduce and the doubling paradigms. It works as follows.

Let the first point in the input stream be its own core-set and call the core-set K_1. Next, as long as the new arriving points lie inside $(1+\varepsilon)\text{MEB}(K_1)$, do nothing. Otherwise, if p_i denotes the new point, call Bădoiu and Clarkson's algorithm on the set $K_1 \cup \{p_i\}$. This gives a new core-set K_2. In general, maintain a sequence of core-sets $\mathcal{K} = \langle K_1, \ldots, K_u \rangle$ and whenever a new point p_i arrives such that it does not lie in $(1+\varepsilon)\text{MEB}(K_j)$ for any j, call Bădoiu and Clarkson's algorithm on the set $\bigcup_{j=1}^{u} K_j \cup \{p_i\}$.

The sequence $\mathcal{K}$ of core-sets is similar to the set of core-sets maintained in the merge-and-reduce algorithm from Section 2.2.2 for computing the kernel in $O(\sqrt{n})$ space, except for one important difference: the core-sets satisfy the property that a set K_i serves as the core-set for the union of all sets K_j for $j < i$. However, this did not happen in the merge-and-reduce algorithm. We will see later that this property of the core-sets is useful for proving an upper bound on the approximation factor without requiring any decomposability properties.

The algorithm is similar to doubling because at any given point, the algorithm maintains a core-set for the points seen till now and updates this set when a new point arrives that is not "served" by this core-set.

The only problem is that the size of the sequence $\mathcal{K}$ might become too large. To reduce space, whenever a new call to the subroutine is made, the algorithm also removes some of the previous K_i's when $r(\text{MEB}(K_i))$ is smaller than $O(\varepsilon)r(\text{MEB}(K_u))$. Agarwal and Sharathkumar proved that this removal process does not hurt the effectiveness of the data structure.

To prove correctness of their algorithm, Agarwal and Sharathkumar showed that the following invariants are maintained throughout the course of the algorithm, where $B_i = \text{MEB}(K_i)$:

(P1) For all i, $r(B_{i+1}) \geq (1 + \Omega(\varepsilon^2))r(B_i)$.

(P2) For all $i < j$, $K_i \subset (1+\varepsilon)B_j$.

(P3) $P \subset \bigcup_{i=1}^{u}(1+\varepsilon)B_i$.

The sequence $\mathcal{K}$ of core-sets was called an *ε-blurred ball cover* in the paper. Property (P1) ensures that the number of core-sets maintained at any time is $u = O(\log(1/\varepsilon))$. Since each core-set has size $O(1/\varepsilon)$, the total space is $O(d)$ for constant ε. Let $B = \text{MEB}(\bigcup_{i=1}^{u} B_i)$ (computable by brute force). Property (P3) ensures that $(1+\varepsilon)B$ encloses P. Using property (P2), Agarwal and Sharathkumar proved that $r(B) \leq (\frac{1+\sqrt{3}}{2} + \varepsilon)\, r(\text{MEB}(P))$, thus giving a factor-1.366 algorithm for MEB in the streaming model.

3.5 An improved analysis

In this section, we show that in fact, the approximation factor for Agarwal and Sharathkumar's algorithm is less than 1.22. The proof amounts to establishing the following (purely geometric) theorem:

Theorem 1. *Let $K_1, \ldots, K_u$ be subsets of a point set P in $\mathbb{R}^d$, with $B_i = \text{MEB}(K_i)$, such that $r(B_i)$ is increasing over i and property* (P2) *is satisfied for a sufficiently small $\varepsilon > 0$. Let $B = \text{MEB}(\bigcup_{i=1}^{u} B_i)$. Then $r(B) < (1.22 + O(\varepsilon))\, r(\text{MEB}(P))$.*

We will prove Theorem 1 in the next few subsections. First we need the following well-known fact, often used in the analysis of high-dimesional MEB algorithms (e.g. see [10]):

Lemma 1 (the "hemisphere property")**.** *Let P be a set of points in $\mathbb{R}^d$. There is no hemisphere of* $\text{MEB}(P)$ *that does not contain a point from P. In other words, assuming the origin to be at the center of* $\text{MEB}(P)$*, for any unit vector v, there exists a point $p \in P$ such that p lies on the boundary of* $\text{MEB}(P)$ *and $v \cdot p \leq 0$.*

We introduce a few notations. Without loss of generality, let $r(B) = 1$ and $c(B)$ be the origin. Let u_i be the unit vector in the direction of the center of B_i and $\sigma_{ij} = u_i \cdot u_j$ be

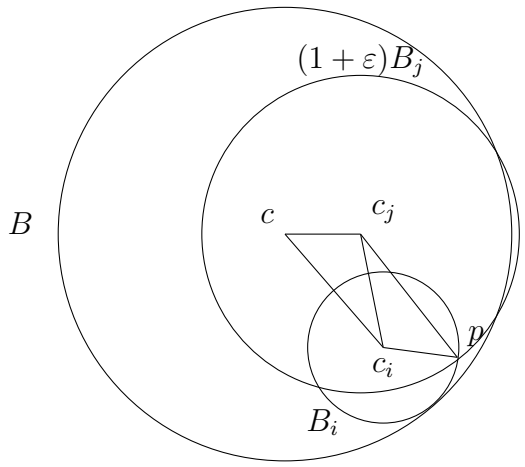

Figure 3.1: Proof of Lemma 2

the inner product between the vectors u_i and u_j. Let us also write $r(B_i)$ simply as r_i and set $t_i = 1/(1-r_i)$. Note that the $t_i \geq 1$ are increasing over i.

Lemma 2. *For all $i < j$ with $t_i \leq t_j < 10$ such that B_i and B_j touch ∂B,*

$$\sigma_{ij} \;\geq\; \frac{t_j}{t_i} - t_j + t_i - O(\varepsilon).$$

Proof. Let c, c_i, c_j be the centers of the balls B, B_i, B_j respectively. Figure 3.5 shows the projection of $B, (1+\varepsilon)B_i, B_j$ onto the plane formed by c, c_i, c_j. Let p be one of the points where $(1+\varepsilon)B_j$ intersects B_i in this plane (let us assume for now that they intersect and remove the assumption later). Applying the cosine law to the triangle c_ic_jc, we get

$$\|c_ic_j\|^2 = \|cc_j\|^2 + \|cc_i\|^2 - 2\|cc_i\|\|cc_j\|\sigma_{ij}. \tag{3.1}$$

Next, we apply the hemisphere property to the ball $B_i = \text{MEB}(K_i)$. Choosing v to be the vector $c_j - c_i$, we deduce the existence of a point $q \in K_i$ such that q lies on ∂B_i and $\angle c_jc_iq \geq \pi/2$. By property (P2) of the blurred ball cover, we know that $q \in K_i \subset (1+\varepsilon)B_j$. Since $\|c_ip\| = \|c_iq\|$ and $\|c_jp\| \geq \|c_jq\|$, we have $\angle c_jc_ip \geq \angle c_jc_iq \geq \pi/2$. This means

$$\|c_jp\|^2 \geq \|c_ic_j\|^2 + \|c_ip\|^2. \tag{3.2}$$

Substituting $\|c_jp\| = (1+\varepsilon)r_j, \|c_ip\| = r_i, \|cc_j\| = 1-r_j, \|cc_i\| = 1-r_i$ into (3.1) and (3.2) and combining them, we get

$$(1+\varepsilon)^2 r_j^2 \;\geq\; (1-r_j)^2 + (1-r_i)^2 - 2(1-r_i)(1-r_j)\sigma_{ij} + r_i^2.$$

Letting $s_i = 1-r_i$ and $s_j = 1-r_j$ and $t_i = 1/s_i$ and $t_j = 1/s_j$, we get

$$\begin{aligned}
(1+\varepsilon)^2(1-2s_j+s_j^2) &\geq s_i^2 + s_j^2 - 2s_is_j\sigma_{ij} + (1-2s_i+s_i^2) \\
\implies \qquad 2s_is_j\sigma_{ij} &\geq 2s_i^2 - 2s_i + 2s_j - O(\varepsilon) \\
\implies \qquad \sigma_{ij} &\geq t_i - t_j + t_j/t_i - O(\varepsilon t_i t_j).
\end{aligned}$$

(The assumption $t_i \leq t_j < 10$ allows us to rewrite $O(\varepsilon t_i t_j)$ as $O(\varepsilon)$.)

Now, in case when B_i and $(1+\varepsilon)B_j$ do not intersect, B_i completely lies inside $(1+\varepsilon)B_j$. Then we can choose p to be the point on the boundary of B_i such that $\angle c_jc_ip = \pi$. Thus inequality (3.2) will still be satisfied. Also, we will have $\|c_jp\| \leq (1+\varepsilon)r_j, \|c_ip\| = r_i, \|cc_j\| = 1-r_j, \|cc_i\| = 1-r_i$. Thus the substitution step will also work. □

3.5.1 Proof of factor 4/3

As a warm-up, in this subsection, we give a short proof of a weaker 4/3 upper bound on the constant in Theorem 1.

Let B_i be the largest ball that touches ∂B. Since B is the minimum enclosing ball of $\bigcup_{\ell=1}^{u} B_\ell$, by applying the hemisphere property to B with $v = u_i$ there must exist another ball B_j such that $\sigma_{ij} \leq 0$. Combining with Lemma 2, we get

$$\frac{t_i}{t_j} - t_i + t_j \;\leq\; O(\varepsilon) \quad\implies\quad t_i \;\geq\; \frac{t_j - O(\varepsilon)}{1 - 1/t_j}.$$

Since $t_j \geq 1$, the minimum value achievable by t_i that satisfies the above inequality can be easily found to be $4 - O(\varepsilon)$ (attained when $t_j \approx 2$). This translates to a minimum value of $3/4 - O(\varepsilon)$ for $r_i = 1 - 1/t_i$. Since $r(\mathrm{MEB}(P)) \geq r_i$ and $r(B) = 1$, this proves a version of Theorem 1 with the constant $4/3 + O(\varepsilon)$.

Remark: We have implicitly assumed that $t_j \leq t_i < 10$ when applying Lemma 2, but this is without loss of generality since $t_i \geq 10$ would imply $r_i > 0.99$ giving an approximation factor of ≈ 1.01.

3.5.2 Proof of factor 16/13

In attempting to find an example where the 4/3 bound might be tight, one could set $t_i = 4$ and $t_j = 2$, which implies $\sigma_{ij} \approx 0$ by Lemma 2, i.e., u_i and u_j are nearly orthogonal. However, by the hemisphere property, B would not be defined by the 2 balls B_i, B_j alone. This suggests that an improved bound may be possible by considering 3 balls instead of just 2, as we will demonstrate next.

Let B_i be the largest ball that touches ∂B, and B_j be the smallest ball that touches ∂B. Let $\alpha \geq 0$ be a parameter to be set later. By applying the hemisphere property to B with $v = u_i + \alpha u_j$, there must exist a k such that B_k touches ∂B and $u_k \cdot (u_i + \alpha u_j) \leq 0$. This means

$$\sigma_{ik} + \alpha \sigma_{jk} \leq 0. \tag{3.3}$$

Note that $t_j \leq t_k \leq t_i$. By Lemma 2, we get

$$\begin{aligned} &\frac{t_i}{t_k} - t_i + t_k + \alpha \left(\frac{t_k}{t_j} - t_k + t_j \right) \leq O(\varepsilon) \\ &\implies t_i \geq \frac{t_k + \alpha(t_k/t_j - t_k + t_j) - O(\varepsilon)}{1 - 1/t_k} \geq \frac{t_k + \alpha(2\sqrt{t_k} - t_k) - O(\varepsilon)}{1 - 1/t_k}. \end{aligned}$$

The last step follows since the minimum of $t_k/x + x$ is achieved when $x = \sqrt{t_k}$ (e.g., by the A.M.–G.M. inequality). The final expression from the last step is in one variable, and can be minimized using standard techniques. Obviously, the minimum value depends on α. As it turns out, the best bound is achieved when $\alpha = 4/3$ and the minimum value is $16/3 - O(\varepsilon)$ (attained when $t_k \approx 4$). Thus, $t_i \geq 16/3 - O(\varepsilon)$, implying $r_i = 1 - 1/t_i \geq 13/16 - O(\varepsilon)$ and an upper bound of $16/13 + O(\varepsilon)$ in Theorem 1.

3.5.3 Proof of factor 1.22

For our final proof of Theorem 1, the essential idea is to consider 4 balls instead of 3.

As before, let B_i be the largest ball that touches ∂B, and B_j be the smallest ball that touches ∂B. Choose a parameter $\alpha = \alpha(t_j) \geq 0$; unlike in the previous subsection, we find that making α dependent on t_j can help. By the hemisphere property, there must exist a B_k that touches ∂B while satisfying (3.3): $\sigma_{ik} + \alpha\sigma_{jk} \leq 0$. By applying the hemisphere property once more with $v = \beta u_i + \gamma u_j + u_k$, for every $\beta, \gamma \geq 0$, there must exist a B_ℓ that touches ∂B satisfying

$$\beta\sigma_{i\ell} + \gamma\sigma_{j\ell} + \sigma_{k\ell} \leq 0. \tag{3.4}$$

We prove that with Lemma 2, these constraints force $t_i > 5.54546$, implying $r_i = 1 - 1/t_i > 0.8197$ and the claimed 1.22 bound in Theorem 1. We need a noticeably more intricate argument now, to cope with this more complicated system of inequalities. Our strategy is to assume $t_i \leq \tau := 5.54546$ and then prove a contradiction.

Note that 2 cases are possible: $t_j \leq t_k \leq t_\ell \leq t_i$ or $t_j \leq t_\ell \leq t_k \leq t_i$. We first eliminate the variable t_ℓ in (3.4). By (3.4), we have $\forall \beta, \gamma \geq 0$:

$$\begin{aligned}&\left[\exists t_\ell \in [t_k, \tau] : \beta\left(\frac{\tau}{t_\ell} - \tau + t_\ell\right) + \gamma\left(\frac{t_\ell}{t_j} - t_\ell + t_j\right) + \frac{t_\ell}{t_k} - t_\ell + t_k \leq O(\varepsilon)\right] \vee \\ &\left[\exists t_\ell \in [t_j, t_k] : \beta\left(\frac{\tau}{t_\ell} - \tau + t_\ell\right) + \gamma\left(\frac{t_\ell}{t_j} - t_\ell + t_j\right) + \frac{t_k}{t_\ell} - t_k + t_\ell \leq O(\varepsilon)\right].\end{aligned} \tag{3.5}$$

Here, the first predicate represents the first case, where t_ℓ lies between t_k and t_i. The second predicate represents the second case, where t_l lies between t_j and t_k.

Observe that in each of the two cases, multiplying the left hand side by t_ℓ yields a quadratic inequality in t_ℓ of the form $at_\ell^2 + bt_\ell + c \leq 0$. (The $O(\varepsilon)$ terms are negligible.) In the first case,

$$a = \beta + \gamma/t_j - \gamma + 1/t_k - 1, b = -\beta\tau + \gamma t_j + t_k, \text{ and } c = \beta\tau. \tag{3.6}$$

And in the second case,

$$a = \beta + \gamma/t_j - \gamma + 1, b = -\beta\tau + \gamma t_j - t_k, \text{ and } c = \beta\tau + t_k. \tag{3.7}$$

The variable t_ℓ can then be eliminated by the following rule:

$$\begin{aligned}
&(\exists x \in [x_1, x_2] : ax^2 + bx + c \leq 0) \text{ iff} \\
&(ax_1^2 + bx_1 + c \leq 0) \vee (ax_2^2 + bx_2 + c \leq 0) \vee \\
&[((a \geq 0) \wedge (b^2 \geq 4ac) \wedge (2ax_1 \leq -b \leq 2ax_2))].
\end{aligned} \tag{3.8}$$

For β, we try two fine-tuned choices: (i) $\beta = -\gamma(\tau/t_j - \tau + t_j) - (\tau/t_k - \tau + t_k) + O(\varepsilon)$ (which is designed to make the above inequality tight at $t_\ell = \tau$), and (ii) a root β of the equation $b^2 = 4ac$ where a, b, c are the coefficients of the first quadratic inequality in the preceding paragraph (for fixed t_j, t_k, γ, this is a quadratic equation in β). As it turns out, these two choices are sufficient to derive the contradiction at the end. Note that the reason for choosing these two values is essentially based on the intuition that to be able to prove a contradiction, one should try to make as many inequalities tight as possible.

Three variables γ, t_j, t_k still remain and the function $\alpha(t_j)$ has yet to be specified. At this point, it is best to switch to a numerical approach. We wrote a short C program (see Appendix A) to perform the needed calculations. For γ, we try a finite number of choices, from 0 to 1 in increments of 0.05, which are sufficient to derive the desired contradiction. For (t_j, t_k), we divide the two-dimensional search space into grid cells of side length 0.0005. For each grid cell that intersects $\{t_k \geq t_j\}$, we lower-bound the coefficients of the above quadratic inequalities over all (t_j, t_k) inside the cell, and attempt to obtain a contradiction with (3.4) by the strategy discussed above. If we substitute the lower bounds on a, b and c into the inequality $at_\ell^2 + bt_\ell + c \leq 0$ and get that it is not satisfied for any t_ℓ, then that means it is not satisfied for any t_ℓ at any point inside the grid cell. This is because of the fact that t_ℓ is positive and therefore the value of $at_\ell^2 + bt_\ell + c$ increases if any one of a, b or c increases.

If we are not able to get a contradiction for the cell this way, we turn to (3.3), which implies

$$\frac{\tau}{t_k} - \tau + t_k + \alpha \left(\frac{t_k}{t_j} - t_k + t_j \right) \leq O(\varepsilon);$$

from this inequality, we can generate an interval of α values that guarantees a contradiction in the (t_j, t_k) cell. We set $\alpha(t_j)$ to any value in the intersection of all α-intervals generated in the grid column of t_j. After checking that the intersection is nonempty for each grid column, the proof is complete.

Remarks: Our analysis of the system of inequalities derived from (3.3) and (3.4) is close to tight, as an example shows that these inequalities cannot yield a constant better than 1.219 regardless of the choice of the function $\alpha(t_j)$: Consider $t_i = 5.56621$ and $t_j = 2$. If $\alpha < 1.15$, pick $t_k = 2.67$; otherwise, $t_k = 5.08$. Now, pick 100 uniformly spaced points from the interval $[t_j, t_k]$ and the interval $[t_k, t_i]$. For each point, let t_ℓ be that value. This gives one linear inequality in terms of β and γ. Thus we have 100 such inequalities in total for the 100 different values of t_ℓ. We want to prove that for all β and γ, one of them is satisfied. We can show this by proving the complement, i.e., there does not exist any value for the pair (β, γ) for which the negation of all the inequalities are satisfied. This is essentially equivalent to proving that a 100 constraint linear program is infeasible, which we have verified using Maple (see Appendix B).

By choosing B_k and B_ℓ more carefully, one could add in the constraints $\sigma_{ij} \leq 0$, $\sigma_{ij} \leq \sigma_{ik}$, $\sigma_{ij} \leq \sigma_{i\ell}$, and $\sigma_{ik} + \alpha\sigma_{jk} \leq \sigma_{i\ell} + \alpha\sigma_{j\ell}$, though $t_j \leq t_k, t_\ell$ is no longer guaranteed; however, the system of inequalities becomes even harder to optimize, and we suspect that any improvements would be very small. Likewise, an analysis involving 5 or more balls does not seem to be worth the effort, until new ideas are found to simplify matters.

Chapter 4

Dynamic MEB

4.1 Preliminaries

In the dynamic setting, we allow points to be inserted and deleted. The task is to answer queries regarding the current point set. For our case, we are concerned with maintaining an approximate minimum enclosing ball of points in $\mathbb{R}^d$.

Dynamic algorithms are related to streaming algorithms because a streaming algorithm for a particular problem already gives a dynamic algorithm for the problem in the insertion-only case. Incorporating deletions is often the difficult part.

However, there is a standard randomization trick that helps in making deletions easy. To demonstrate the trick, we convert the simple factor-2 streaming algorithm from Section 3.1 into a factor-2 dynamic algorithm. In the preprocessing stage, pick any random point p_0 from the point set P uniformly and arrange the rest of the points in a priority queue with the key being the distance of the point from p_0. Call p_0 the "anchor point." To insert a new point, simply insert it into the priority queue. This takes time $O(\log n)$, where n is the number of points. The MEB returned at any time is the ball centered at p_0 and having a radius equal to the maximum key. To delete a point, remove it from the priority

queue if the point being deleted is not the anchor point itself. Otherwise, rebuild the whole data structure by picking a new random anchor point p and arranging the rest in a priority queue. Since the choice of the anchor point is random, the probability with which it will be deleted is $1/n$. Therefore the expected cost of deletion is $\frac{1}{n}O(n \log n)+O(\log n) = O(\log n)$. The space used is linear.

A similar randomization trick was used by Chan [12] to get a dynamic algorithm for maintaining ε-kernels in low dimensions. To get a better ratio for high-dimensional minimum enclosing ball, we modify his algorithm in suitable ways. First, we outline his algorithm.

The starting point is a simple constant-factor approximation algorithm for the minimum bounding box [2, 8]. Pick a point $p_0 \in P$. This is the first anchor point. Next, let p_1 be the point farthest from p_0 in P. In general, pick point p_j to be the point farthest from $\operatorname{aff}\{p_0, \ldots, p_{j-1}\}$, where $\operatorname{aff} S$ denotes the affine hull of a set S. The resulting anchor points $p_0, \ldots, p_d$ form a core-set whose minimum bounding box approximates the minimum bounding box of P to within $O(1)$ factor. The factor can be reduced to $1 + \varepsilon$ by building a grid along a coordinate system determined by the anchor points; the size of the core-set increases to $O(\varepsilon^{-d})$.

Now, to make this algorithm dynamic, the approach is to choose the anchor points in some random way and then whenever an anchor point is deleted, rebuild the whole data structure. Because of the randomness, the deleted point will be an anchor point with only a low probability. Thus instead of choosing p_j to be the point farthest from $\operatorname{aff}\{p_0, \ldots, p_{j-1}\}$, we pick p_j uniformly at random from the set A_j of $\alpha|P|$ farthest points from $\operatorname{aff}\{p_0, \ldots, p_{j-1}\}$ and discard A_j. Thus, after picking all the anchor points, we obtain a set $R = \bigcup_j A_j$ of all discarded points. Since R is not "served" by the anchor points chosen, we recurse on R. Since $|R|$ is a fraction less than $|P|$ if the constant α is sufficiently small, this gives us a collection of $O(\log n)$ core-sets. The final core-set returned is the union of all of them. Insertions can be incorporated in a standard way, analogous to the *logarithmic method* by Bentley and Saxe [9].

The transition from Chan's dynamic algorithm for ε-kernels to our dynamic algorithm for minimum enclosing balls in high dimensions is similar to the transition from the merge-and-reduce algorithm for maintaining ε-kernels in the streaming model to Agarwal and Sharathkumar's [4] streaming algorithm for minimum enclosing balls in high dimensions. The problem with directly generalizing the merge-and-reduce algorithm to high dimensions by replacing ε-kernels with Bădoiu and Clarkson's core-sets is that their core-sets are not decomposable. Agarwal and Sharathkumar fixed this issue by making the core-sets "cumulative", i.e., their algorithm makes sure that the i^{th} core-set K_i is a core-set for all K_j's, $j < i$. Replacing the ε-kernels in Chan's algorithm with Bădoiu and Clarkson's core-sets will not work for the same reason. We fix that by using essentially Agarwal and Sharathkumar's technique. The precise details are given in the following paragraphs.

4.2 A new dynamic algorithm

Let P be the set of points whose MEB we want to maintain.

To mimic Chan's randomization trick in the preprocessing stage, we need to make Bădoiu and Clarkson's core-sets randomized in some sense (see Section 3.2 on Bădoiu and Clarkson's algorithm). The first point of the core-set is some arbitrarily chosen point $p_0 \in P$ and the next one is now a point picked uniformly at random from the $\alpha|P|$ farthest points from p in P and so on. Since the core-set thus formed will not "serve" the $\alpha|P|$ farthest points, we collect all those discarded points into the set R and recurse on it. In the end, we get a collection of core-sets from each level of recursion. Unfortunately, the union of all these core-sets is not a core-set of the whole set because of the argument in Section 3.3. Thus instead of recursing just on R, we recurse on the union of R and the core-sets from all the previous levels of recursion as shown in Algorithm 1.

Let P_i be the point set on which we recurse at level i of the recursion. Thus the original set is P_0 and the final set is P_u where u is the number of levels in the recursion. For each set P_i, we associate with it a subset Q_i of P_i of size a constant fraction of the size of P_i. Let

Algorithm 1 P.preprocess()

if $|P| < c \log n$ **then**
  $K_P \leftarrow P$ and return
end if
$Q \leftarrow P$
$p_0 \leftarrow$ random point of P
for $j = 1, \ldots, \lceil 2/\varepsilon \rceil$ **do**
  $A_j \leftarrow$ the $\alpha|P|$ farthest points of Q from $c(\text{MEB}(\{p_0, \ldots, p_{j-1}\}))$
  $Q \leftarrow Q - A_j$
  $p_j \leftarrow$ a random point of A_j
end for
$K_P \leftarrow \{p_0, \ldots, p_{\lceil 2/\varepsilon \rceil}\}$ {an ε-core-set of Q by Bădoiu and Clarkson}
$\widehat{K}_R \leftarrow \widehat{K}_P \cup K_P$ {$\widehat{K}_P$ is a union of core-sets at earlier levels}
$R \leftarrow (P - Q) \cup \widehat{K}_P$ {remember to add earlier core-sets $\widehat{K}_P$ to the next level}
R.preprocess()
P.counter $\leftarrow \delta|P|$

Algorithm 2 P.delete(p), where $p \in P - \widehat{K}_P$

if $|P| < c \log n$ **then**
  remove p from P, reset $K_P \leftarrow P$, and return
end if
remove p from P
P.counter $\leftarrow P$.counter $- 1$
if P.counter $= 0$ or $p \in K_P$ **then**
  P.preprocess() {rebuild all sets after current level}
end if
if $p \in R$ **then**
  R.delete(p)
end if

Algorithm 3 P.insert(p)

if $|P| < c \log n$ **then**
  insert p into P, reset $K_P \leftarrow P$, and return
end if
insert p into P
P.counter $\leftarrow P$.counter $- 1$
if P.counter $= 0$ **then**
  P.preprocess() {rebuild all sets after current level}
end if
R.insert(p)

K_i be Bădoiu and Clarkson's core-set for the set Q_i. Note that K_u, being the base case, is just the whole point set P_u. Also, the set of points in P_i that are not in Q_i constitutes P_{i+1}.

Let $\mathcal{K} = \langle K_1, \ldots, K_u \rangle$ denote the sequence of core-sets K_i over all currently active point sets P, arranged from the root level to the last level. Let $B_i = \text{MEB}(K_i)$. Then property (P3) is satisfied because of Bădoiu and Clarkson's algorithm. The trick of inserting core-sets at earlier levels to the current set ensures property (P2). We can then use Theorem 1 to infer that $B = \text{MEB}(\bigcup_{i=1}^{u} B_i)$ is a 1.22-approximation to MEB(P) for a sufficiently small ε.

To ensure that the algorithm is correct, we maintain properties (P2) and (P3) over insertions and deletions. To ensure that the data structure is of linear size, we maintain the following invariant: there are $O(\log n)$ levels in the data structure with the size of each level at most a constant fraction of the size of the previous level, i.e., $u = O(\log n)$ and $|P_i| = O(n/c'^i)$ for some constant c'.

For insertions, we add the point p to P_i at all levels, except, at level u, we also add it to K_u and Q_u. Thus a newly added point is served by the core-set K_u. For deletions, we delete p from the level where p lies in Q_i and all the levels above it. This makes sure that

(P2) and (P3) are maintained. However, if many updates are made at the same level, then the second invariant may not hold true. Thus we use a counter at each level that counts the number of updates made at that level and we rebuild the whole data structure starting at that level if the counter reaches 0.

Also, instead of applying an exact algorithm to compute B, it is better to first compute a $(1+\varepsilon)$-approximation B_i' to $\text{MEB}(K_i)$ for each i, and then return a $(1+\varepsilon)$-approximation to $\text{MEB}(\bigcup_{i=1}^{u} B_i')$. (Note that every K_i has size $O(1/\varepsilon)$, except for the last set, which has size $O(\log n)$.) The latter can be done by a known approximation algorithm of Kumar, Mitchell, and Yildirim [21], which generalizes Bădoiu and Clarkson's algorithm for sets of balls. The time required is $O(du) = O(d \log n)$ (we ignore dependencies on ε from now on). It can be checked that the proof of Theorem 1 still goes through with B_i replaced by B_i', since the hemisphere property is still satisfied "approximately" for B_i'.

Let $\widehat{K}_i = K_1 \cup \ldots \cup K_i$. Note that for all i, $|\widehat{K}_i| \leq O(u/\varepsilon)$. For $|P_i| \gg u/\varepsilon$, note that $|P_{i+1}|$ is a fraction less than $|P_i|$ if we make the constants α and δ sufficiently small (relative to ε). Thus, for c sufficiently large, u is bounded logarithmically in n.

The **for** loop in P.preprocess() takes $O(dn)$ time for constant ε. Thus, the total preprocessing time is bounded by a geometric series summing to $O(dn)$. Space is $O(dn)$ as well. In the pseudocode for P.delete(), although the cost of the call to P.preprocess() is $O(d|P|)$, it can be shown [12] that the probability of deleting an anchor point $p \in K_i$ is $O(1/|P|)$ at any fixed level. Excluding the cost of computing B, the analysis of the expected amortized update time is essentially the same as in Chan's paper [12] and yields $O(d \log n)$. (The randomized analysis assumes that the update sequence is oblivious to the random choices made by the algorithm.) We conclude:

Theorem 2. *A factor-*1.22 *approximation of the* MEB *of points in* $\mathbb{R}^d$ *can be maintained with an algorithm that takes preprocessing time* $O(dn \log n)$*, uses space* $O(dn)$*, and takes expected amortized time* $O(d \log n)$ *for the updates.*

Chapter 5

Miscellaneous Results and Final Remarks

5.1 Multiple-pass streaming algorithms

So far we have discussed the model of streaming where only one pass is allowed over the input. It is interesting to consider more than one pass. For example, we saw in Section 2.2.3 that if we are allowed two passes, the minimum enclosing cylinder of a set of points in high dimensions can be found within an approximation factor of 4; however, for one pass we could get only a factor-18 algorithm[1].

For minimum enclosing balls, Bădoiu and Clarkson's algorithm for high-dimensional core-sets can be viewed as a multiple-pass streaming algorithm. In particular, it finds a $(1+\varepsilon)$ approximation to the minimum enclosing ball in $O(1/\varepsilon)$ passes over the input stream. A more careful analysis shows that $\lceil \frac{2}{\varepsilon} \rceil$ passes are sufficient. In this section, we show that this can be improved, i.e., it is possible to get an approximation factor of $1+\varepsilon$ with fewer passes.

[1]Chan improved the factor-18 to factor-5 using a more intricate algorithm.

Consider the following multi-pass streaming algorithm. Run Agarwal and Sharathkumar's algorithm in the first pass. In the end, we will have the sequence $\mathcal{K} = \langle K_1, \ldots K_u \rangle$ of core-sets. Next, delete all points from memory except for the points in K_u. Let $x = t_u$ and re-initialize all t_i's. In the second pass, run the same algorithm again, but assume that points in K_u have arrived before all other points in the stream.

To see why this gives a better approx. factor at the end of the second pass, we can do another three-ball analysis similar to the one done in Section 3.5.2. If we use $\alpha = 1$ in (3.3), we get $\sigma_{ik} + \alpha\sigma_{jk} \leq O(\delta)$, which gives $t_i/t_k - t_i + t_k + t_k/t_j - t_k + t_j \leq 0$ (recalling that B_i is the largest ball and B_j is the smallest)[2]. Since t_i/t_k and t_k/t_j are both at least 1, we get $2 - t_i + t_j \leq O(\delta)$, which finally gives us $t_i \geq t_j + 2 - O(\delta)$. Thus, after each pass, the value of t_i increases by at least $2 - O(\delta)$. The approximation factor is given by $\frac{1}{1-1/t_i}$. To get a factor of $1 + \varepsilon$, we need t_i to be $\approx \frac{1}{\varepsilon}$. Thus, $\frac{0.5}{\varepsilon} + O(\delta)$ passes suffice.

Theorem 3. *A $(1+\varepsilon)$-factor approximation to the minimum enclosing ball of n points in $\mathbb{R}^d$ can be obtained in $\frac{0.5+\delta}{\varepsilon}$ passes over the input, for an arbitrarily small constant $\delta > 0$.*

This can perhaps be improved by doing an analysis with more balls, which we leave as an open problem.

5.2 Open problems

Of course, the main question that remains to be answered is to determine the best approximation factor possible for minimum enclosing balls in high dimensions in the streaming model. Agarwal and Sharathkumar gave a lower bound of $(1 + \sqrt{2})/2 \approx 1.207$. Although our result reduces the gap between the upper and the lower bound, it does not completely eliminate it. Agarwal and Sharathkumar also gave an example where their algorithm performs strictly worse than the lower bound. Thus either the lower bound is not tight or we need a new algorithm.

[2]We rename ε in Chapter 3 as δ

There are no lower bounds known if we allow more than one pass over the input. The lower-bound proof technique by Agarwal and Sharathkumar already fails to work when we allow two passes. As such, we do not know how far the algorithm from Section 5.1 is from the optimum.

On the other hand, there are some lower bounds on the closely related problem of linear programming in fixed dimensions. For example, Chan and Chen [13] proved that for any fixed dimension, solving a linear program in p passes requires space $\Omega(n^{1/p})$ in a restricted model where the only objects we are allowed to store are indices and points from the input. Guha and McGregor [18] later proved the same bound for the general computational model on the number of bits of space. However, these lower bounds are only on the exact algorithms and provide a trade-off between the number of passes and space required. No lower bound is known that provides trade-offs between the approximation factor and the number of passes.

In the dynamic case, the best ratio achievable with a data structure that requires a polylogarithmic update time and linear space is not known. Because of the lack of a lower bound, even a $(1+\varepsilon)$-factor algorithm is not ruled out.

APPENDICES

Appendix A

C code for proving a factor of 1.22

```
#include <stdio.h>
#include <math.h>
#define MAX(X,Y)  (((X)>(Y))?(X):(Y))

//d is used for defining the grid size of (t_j, t_k).
//dd is used for the size of intervals used for gamma.

//Constants with 'min' or 'max' appended at the end
//are used to specify the range for the corresponding
//variables. Indeed, t_i is an upper bound for both
//t_j and t_k and 1 is a lower bound.

//For gamma, we try values between 0 and 1.

const double t_i = 5.54546,
  d = 0.0005, dd = 0.05, delta = 1e-8,
  t_j_min = 1.0, t_j_max = t_i, t_k_min = 1.0, t_k_max = t_j,
```

```
  gamma_min = 0., gamma_max = 1.;

//t_j_m and t_k_m will denote rounded up values
//of t_j and t_k.

double t_j, t_k, t_j_m, t_k_m, gamma;

//This function checks whether there exists a value x
//between u and v such that the quadratic inequality
//a*x*x+b*x+c<=0 is satisfied. Returns 1 if the
//answer is yes, 0 otherwise. See (3.8).

int check_parabola(double a, double b, double c, double u, double v) {
  return a*u*u+b*u+c <= 0 || a*v*v+b*v+c <= 0 ||
         (a>=0 && b*b-4*a*c>=0 && 2*a*u+b<=0 && 2*a*v+b>=0);
}

//This function checks if (3.5) is satisfied
//for some value of t_l, given a particular value of beta
//and a particular value of gamma. It returns 1 if it's satisfied,
//otherwise it returns 0. We use (3.6) and (3.7) and check_parabola().

int check(double beta) {
  return (beta < 0) || (gamma < 0) ||
         check_parabola(beta+gamma/t_j_m-gamma+1/t_k_m-1,
                        -beta*t_i+gamma*t_j+t_k,
                        beta*t_i, t_k, t_i) ||
         check_parabola(beta+gamma/t_j_m-gamma+1,
                        -beta*t_i+gamma*t_j-t_k_m,
```

```
                        beta*t_i+t_k, t_j, t_k_m);
}

main()
{
  double alpha_low, alpha_high, t_k_low, t_k_high;

  //The next two for loops divide the space of (t_j, t_k)
  //into a grid with side length d.

  for (t_j = t_j_min; t_j < t_j_max; t_j+=d) {
    alpha_low = 0; alpha_high = 1e8;

    //As explained in Section 3.5.3, we need to get
    //contradictions only for the case when t_k>=t_j.
    //Thus, we start the for loop at t_k = t_j - d.

    for (t_k = t_j-d; t_k < t_k_max; t_k+=d) {

      //t_j_m and t_k_m are the rounded up values
      //of t_j and t_k for the particular grid cell.

      t_j_m = t_j+d; t_k_m = t_k+d;
      int flag = 1;

      //The variable flag maintains whether we have found
      //a contradiction or not. If we haven't found a
      //contradiction, flag remains set to 1, else it
      //becomes 0.
```

```
//First attempt at contradiction.
//Once we are in a particular grid cell for
//(t_j, t_k), we look at various values of gamma.

for (gamma = gamma_min; gamma <= gamma_max; gamma+=dd) {

  //We first try the first fine-tuned value for beta.
  //Note that we use the rounded up values of t_j and t_k
  //at certain places to make sure that the statement
  //we are trying to make holds for all values inside
  //the grid cell.

  flag = flag && check(MAX(-(gamma*(t_i/t_j_m-t_i+t_j)
                       +t_i/t_k_m-t_i+t_k),0) + delta);

  //Next, we try the second fine-tuned value for beta.
  //We solve the quadratic equation by finding the
  //co-efficients of beta^2, beta and the constant term
  //and then using the quadratic formula.

  double a = t_i*t_i-4*t_i,
         b = -2*t_i*(C*t_j+t_k)-4*t_i*(gamma/t_j_m-gamma+1/t_k_m-1),
         c = (gamma*t_j+t_k)*(gamma*t_j+t_k);
  if (b*b>4*a*c)
    flag =  flag && check((-b - sqrt(b*b-4*a*c))/(2*a) + delta);
}

//If there is still no contradiction, then we resort
```

```
      //to the second method, i.e., find a suitable alpha(t_j).
      //We do that by maintaining the range of values of
      //alpha that gives a contradiction. The range is maintained
      //using the variables alpha_low and alpha_high. In the end,
      //there exists a suitable value of alpha if and only if
      //alpha_low <= alpha_high.

      if (flag) {
        double x = -(t_i/t_k_m-t_i+t_k)/(t_k/t_j_m-t_k_m+t_j);
        if (t_k/t_j_m-t_k_m+t_j >= 0 && x > alpha_low)
          { alpha_low = x; t_k_low = t_k; }
        else if (t_k/t_j_m-t_k_m+t_j < 0 && x < alpha_high)
          { alpha_high = x; t_k_high = t_k; }
      }
    } // end t_k for loop

    if (alpha_low > alpha_high)
      printf("error: %f %f %f (%f %f)\n", t_j, t_k_low,
                       t_k_high, alpha_low, alpha_high);
  } // end t_j for loop
}
```

The theorem in Section 3.5.3 follows from the fact that when we run this code, it does not execute the `printf` statement in the end.

Appendix B

Maple code for proving that doing better than 1.219 needs new ideas

See Remarks at the end of Section 3.5.3.

```
with(simplex):
n := 50;
t_i := 5.56621;
t_j := 2;
t_k := 2.67;
alpha := 1.15;
t_k/t_j-t_k+t_j;  # this value should be positive
t_i/t_k-t_i+t_k+alpha*(t_k/t_j-t_k+t_j);  # this value should be negative
minimize(beta, {
  seq(beta*(t_i/t_l-t_i+t_l)
            +gamma*(t_l/t_j-t_l+t_j)
            +t_l/t_k-t_l+t_k >= 0,
            t_l=t_k..t_i, (t_i-t_k)/n),
  seq(beta*(t_i/t_l-t_i+t_l)
```

```
            +gamma*(t_l/t_j-t_l+t_j)
            +t_k/t_l-t_k+t_l >= 0,
            t_l=t_j..t_k,
            (t_k-t_j)/n),
  beta >= 0, gamma >= 0 });  # check that this LP isinfeasible
t_k := 5.08;
t_k/t_j-t_k+t_j;  # this value should be negative
t_i/t_k-t_i+t_k+alpha*(t_k/t_j-t_k+t_j);  # this value should be negative
minimize(beta, {
  seq(beta*(t_i/t_l-t_i+t_l)
      +gamma*(t_l/t_j-t_l+t_j)
      +t_l/t_k-t_l+t_k >= 0,
      t_l=t_k..t_i,
      (t_i-t_k)/n),
  seq(beta*(t_i/t_l-t_i+t_l)
      +gamma*(t_l/t_j-t_l+t_j)
      +t_k/t_l-t_k+t_l >= 0,
      t_l=t_j..t_k,
      (t_k-t_j)/n),
  beta >= 0, gamma >= 0 });  # check that this LP is infeasible
```

References

[1] Designing a digital future: Federally funded research and development in networking and information technology, `http://www.whitehouse.gov/sites/default/files/microsites/ostp/pcast-nitrd-report-2010.pdf`. *Report to the President and Congress*, 2010.

[2] Pankaj K. Agarwal, Sariel Har-Peled, and Kasturi R. Varadarajan. Approximating extent measures of points. *Journal of the ACM*, 51:139–186, 2004.

[3] Pankaj K. Agarwal, Sariel Har-Peled, and Kasturi R. Varadarajan. Geometric approximation via coresets. In Janos Pach and Emo Welzl, editors, *Combinatorial and Computational Geometry, MSRI*, pages 1–30. Cambridge University Press, 2005.

[4] Pankaj K. Agarwal and R. Sharathkumar. Streaming algorithms for extent problems in high dimensions. In *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1481–1489, 2010.

[5] Charu C Aggarwal. *Data streams: models and algorithms*. Advances in Database Systems. Springer, New York, NY, 2007.

[6] 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, 1999.

[7] Amitabha Bagchi, Amitabh Chaudhary, David Eppstein, and Michael T. Goodrich. Deterministic sampling and range counting in geometric data streams. *ACM Trans. Algorithms*, 3, May 2007.

[8] G. Barequet and S. Har-Peled. Efficiently approximating the minimum-volume bounding box of a point set in three dimensions. *Journal of Algorithms*, 38:91–109, 2001.

[9] J. L. Bentley and J. B. Saxe. Decomposable searching problems I: Static-to-dynamic transformations. *Journal of Algorithms*, 1:301–358, 1980.

[10] Mihai Bădoiu and Kenneth L. Clarkson. Optimal core-sets for balls. *Computational Geometry Theory and Applications*, 40:14–22, May 2008.

[11] Timothy M. Chan. Faster core-set constructions and data-stream algorithms in fixed dimensions. *Comput. Geom. Theory Appl.*, 35:20–35, August 2006.

[12] Timothy M. Chan. Dynamic coresets. In *Proceedings of the 24th Annual Symposium on Computational Geometry*, pages 1–9, 2008.

[13] Timothy M. Chan and Eric Y. Chen. Multi-pass geometric algorithms. *Discrete and Computational Geometry*, 37:79–102, January 2007.

[14] Yi Chiang and Roberto Tamassia. Dynamic algorithms in computational geometry. Technical report, 1991.

[15] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer., 3 edition, 2008.

[16] Philippe Flajolet. Approximate counting: A detailed analysis. *BIT*, 25:113–134, 1985.

[17] Philippe Flajolet and G. Nigel Martin. Probabilistic counting. In *Proceedings of the 24th IEEE Symposium on Foundations of Computer Science*, pages 76 –82, 1983.

[18] Sudipto Guha and Andrew McGregor. Tight lower bounds for multi-pass stream computation via pass elimination. In *Proceedings of the 35th International Colloquium on Automata, Languages and Programming, Part I*, pages 760–772, 2008.

[19] Piotr Indyk. Better algorithms for high-dimensional proximity problems via asymmetric embeddings. In *Proceedings of 14th ACM-SIAM Symposium on Discrete Algorithms*, pages 539–545, 2003.

[20] Piotr Indyk and David Woodruff. Optimal approximations of the frequency moments of data streams. In *Proceedings of the 37th ACM Symposium on Theory of Computing*, pages 202–208, 2005.

[21] P. Kumar, J. S. B. Mitchell, and E. A. Yildirim. Approximating minimum enclosing balls in high dimensions using core-sets. *ACM Journal of Experimental Algorithmics*, 8:1.1, 2003.

[22] Muthu Muthukrishnan. *Data Streams: Algorithms and Applications*, volume 1 of *Foundations and Trends in Theoretical Computer Science*. Now Publishers Inc., 2005.

[23] Franco P. Preparata and Michael Ian Shamos. *Computational Geometry: An Introduction*. Springer., 1985.

[24] V. N. Vapnik and A. Ya. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16(2):264–280, 1971.

[25] Jeffrey Scott Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software*, 11:37–57, 1985.

[26] Wikipedia. Earthquake prediction — wikipedia, the free encyclopedia, 2011. [Online; accessed 2-July-2011].

[27] Hai Yu, Pankaj Agarwal, Raghunath Poreddy, and Kasturi Varadarajan. Practical methods for shape fitting and kinetic data structures using coresets. *Algorithmica*, 52:378–402, 2008.

[28] Hamid Zarrabi-Zadeh. An almost space-optimal streaming algorithm for coresets in fixed dimensions. In *Proceedings of 16th European Symposium on Algorithms*, volume 5193 of *Lect. Notes in Comput. Sci.*, pages 817–829. Springer-Verlag, 2008.