Analysis and Implementations of Streaming Algorithms in Computational Geometry

A final report submitted in fulfillment of the thesis adhering to the requirements for the degree of

Bachelor of Technology

By

Shrinivas Acharya

(Roll No. 10010164)

Under the guidance of

Dr. S. V. Rao

Department of Computer Science and Engineering,
Indian Institute of Technology, Guwahati.

April 2014

CERTIFICATE

It is certified that the work contained in the project report titled “Analysis and Implementations of Streaming Algorithms in Computational Geometr”, by Shrinivas Acharya (10010164) has been carried out under my supervision and that this work has not been submitted elsewhere for the award of a degree.

20 April 2014

Dr. S. V. Rao
Professor
Department of Computer Science and Engineering
Indian Institute of Technology Guwahati

DECLARATION

I declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

20 April 2014 Shrinivas Acharya

Dedicated To Everybody

ACKNOWLEDGEMENT

I feel a great privilege in expressing my deepest and most sincere gratitude to my supervisor, Dr. S. V. Rao for the most valuable guidance and influential mentorship provided to me during the course of this project. Due to his technical advices, exemplary guidance, persistent monitoring and constant encouragement throughout the course of my project work, I was able to complete the project through various stages.

I take this opportunity to express my profound gratitude and deep regards to all other people who helped me in every possible way to get my project comfortably completed. I am obliged for the valuable information provided by them in their respective fields. I am grateful for their cooperation during the period of our assignment. Thanks to all of them.

20 April 2014 Shrinivas Acharya

ABSTRACT

In this paper we present the analysis of some fundamental external memory algorithms. We discuss some streaming algorithms in graphs. We present the three models of streaming algorithms named as Semi streaming, W-stream, Stream-Sort. We implemented these algorithms in C and provided the experimental conformations.

In the end we discuss the approximate streaming algorithm for computing the minimum enclosing ball for a given set of points in any dimension (particularly higher dimensions), its implementation in C and provided the experimental conformations.

Chapter 1

Semi Streaming

The algorithm [FKM+05b] is given $\boldsymbol{\Theta(n\ polylog\ n)}$ space where n is the number of vertices in the graph. In this case, the algorithm has enough internal memory to store the vertices but not necessarily the edges in the graph

Minimum Spanning Tree

For computing the minimum spanning tree for a graph, maintaining the connected components can also be used ([FKM+05a]). This algorithm is a streaming version of an algorithm which appears as a remark in [T83].As we read the edges from the stream, we keep track of the connected components in memory. For each connected component, we also maintain a minimum spanning tree (MST). The complete algorithm is as follows:

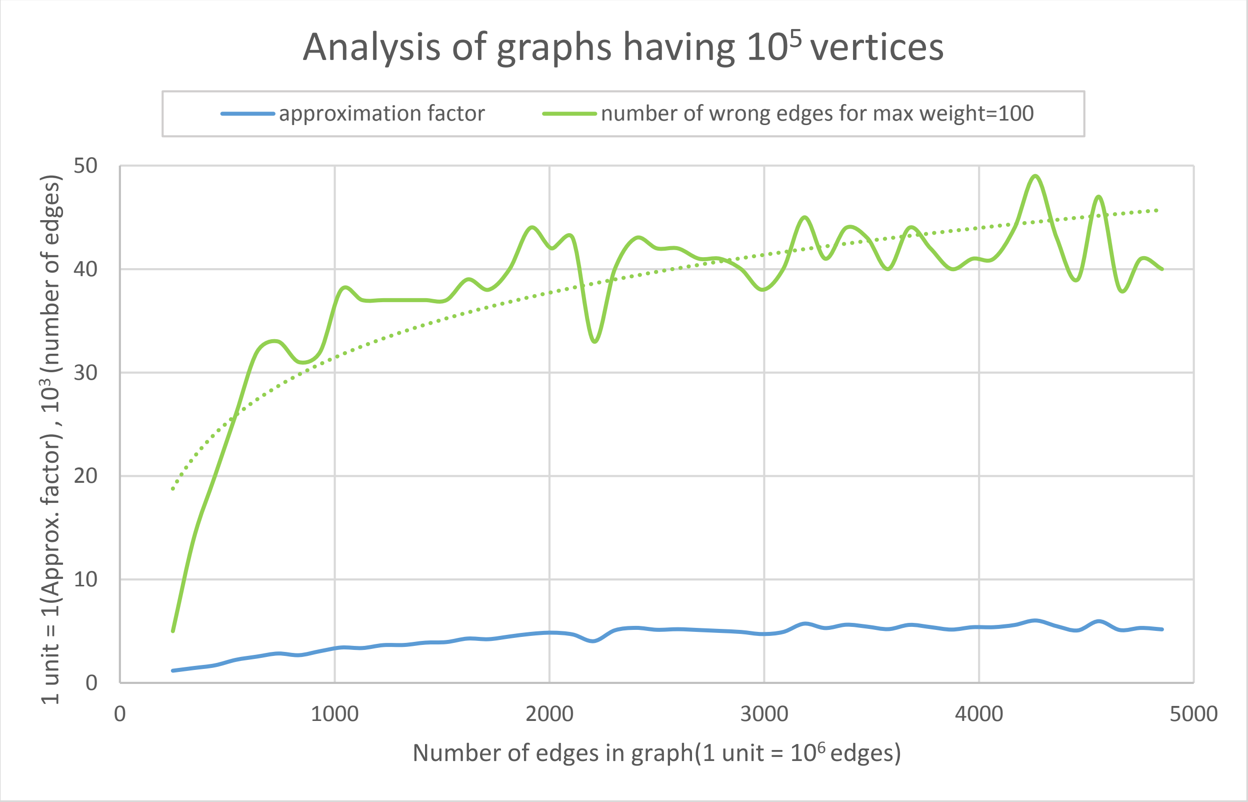
The Algorithm

```
For each edge $(\boldsymbol{u}, \boldsymbol{v})$ in the stream
    if $\boldsymbol{u}$ and $\boldsymbol{v}$ are in different components, do
        union the two components and create a minimum spanning tree for this
        new larger component by merging the two components' minimum
        spanning trees and adding edge$(\boldsymbol{u}, \boldsymbol{v})$.
    else do
        add $(\boldsymbol{u}, \boldsymbol{v})$ to the MST for the component (creating a cycle) and remove
        the heaviest edge on the cycle created.
If the graph is connected, only one component remains. Return its spanning tree as the
result. Otherwise return the individual spanning trees of different components.
```

Clearly number of edges in spanning tree are $\mathrm{O}(n)$. So it takes $\mathrm{O}(n\ ploylog\ n)$space which meets the criteria for semi-streaming model.

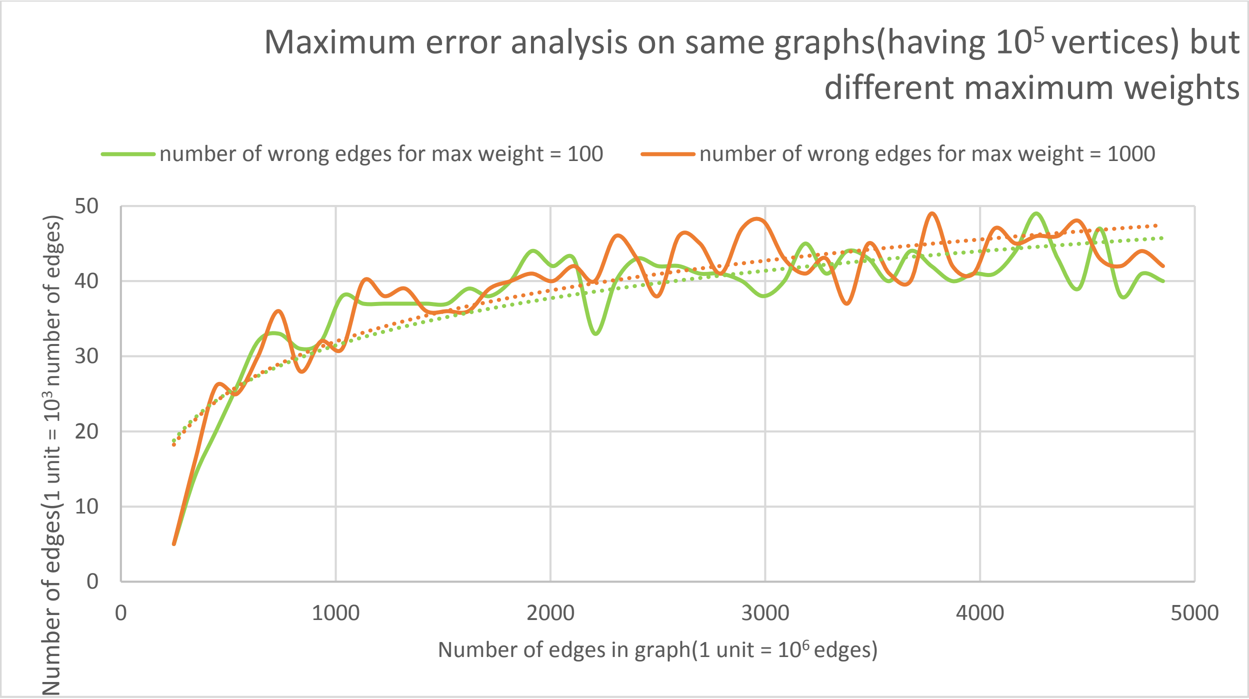
Implementation and Results

Above algorithm was efficiently implemented and results are obtained. Approximation factor was taken by comparing its results with Prim's algorithm. Analysis of results is:



As we can see that approximation factor is increasing with number of edges and reaches as high as 5. But approximation factor is not a good criteria to analyze the results because these results are for random graphs having 10^5 vertices where maximum weight can be 100 and minimum weight can be 1. So if a wrong edge is picked, it would add so much of extra weight. Hence we analyze number of edges that differ from optimal minimum spanning tree (in this case Prim's algorithm) and call it error.

So error is increasing with number of edges and it follows same pattern regardless the values of maximum and minimum weights or their differences. We can see the comparison when maximum weight is 100 and when maximum weight is 1000



Approximation factor increases to very high values in case of 1000 but error is similar in both cases

Chapter 2

W-streaming

In this model ([R03]) the algorithm can write an intermediate stream as it reads the stream. This intermediate stream can be at most a constant factor larger than the original and is used as the input stream for the next pass.

A deterministic W-Stream algorithm for finding the connected components of an undirected that uses s space and $\mathrm{O}((n \log n)/s)$ passes is presented in [DFR06]. It is an open question whether this W-Stream algorithm for connected components can be extended to find a minimum tree as was done with the Semi-Streaming algorithm for connected components.

Chapter 3

Stream-sort Model

This is a newer model for streaming algorithms proposed keeping in mind the efficiency of hardware. As we can sort large set of data efficiently with today's hardware ([V01]), adding a sorting primitive to the W-stream model seems appropriate. This leads to the Stream-Sort model. In each pass through the data, we can either produce an intermediate stream as in the W-Stream model or sort the stream according to some partial order on the items in the stream.

Bound for Stream-sort model is $O(polylog\, n)$ meaning in [R03] and [ADR+04], it is suggested that an algorithm in the Stream-Sort model be considered efficient if the number of streaming and sorting passes is $O(polylog\, n)$. Efficient Stream-Sort graph algorithms for undirected s-t-connectivity, directed s-t-connectivity, minimum spanning tree, maximal independent set, detecting cycles in an undirected graph, and minimum cut appear in [R03] and [ADR+04]. Each of the algorithms is randomized. As with the models discussed in the previous sections, we present a Stream-Sort algorithm for connected components, which is from [R03] and [ADR+04].

Connected Components in Stream Sort

The Algorithm

```
Until there are no more edges in the graph, do
    Assign a random 3 log n bit integer to each vertex in the graph.
    Label each vertex with the minimum number among those assigned to itself and
    its neighbors.
    Merge all vertices that receive the same label.
    If a representative of a connected component is merged then update the
    representative for all vertices in the corresponding component.
```

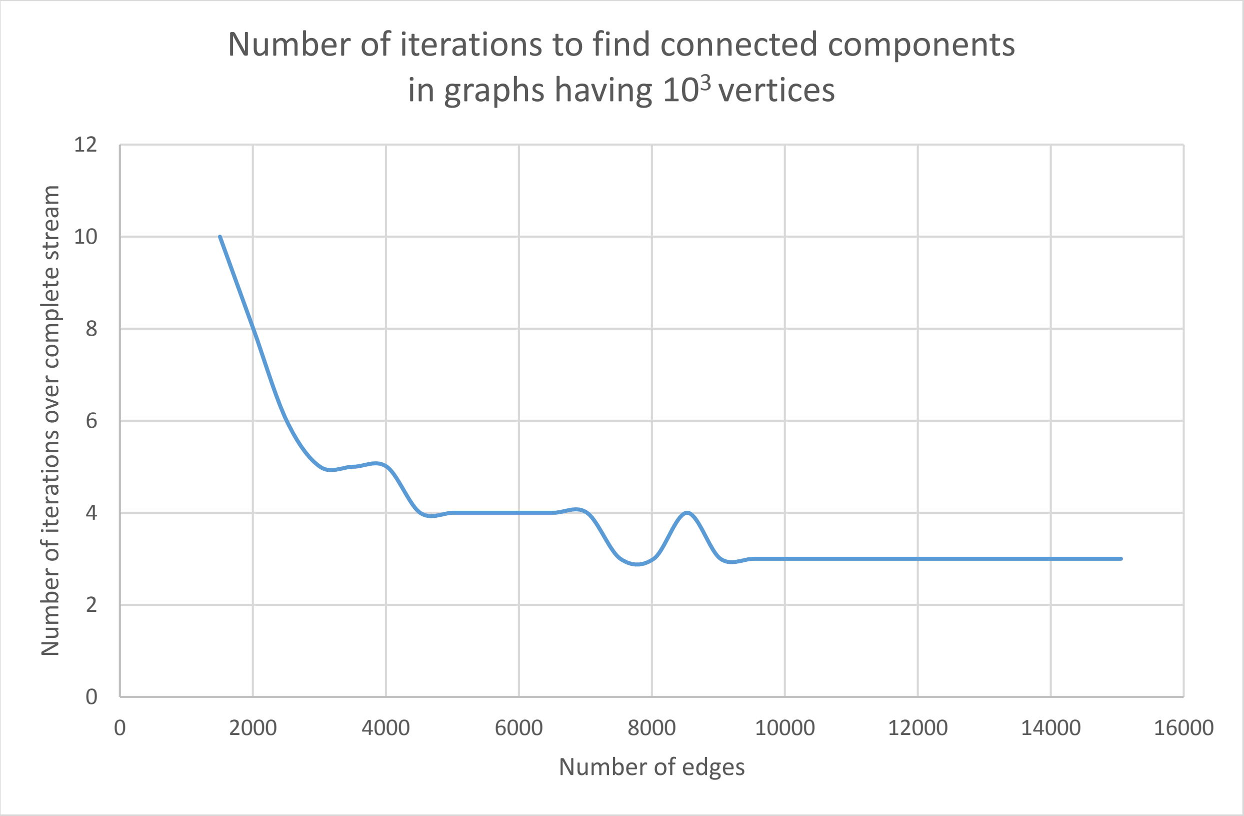
By assigning random labels to each vertex and merging vertices, the number of vertices in the graph decreases during each iteration and expected no of iteration, because of using randomization, comes out to be $O(log\, n)$ ([ADR+04]).

Here input stream is a vertex stream followed by an edge stream. The edge stream must have both $(\boldsymbol{u}, \boldsymbol{v})$ and $(\boldsymbol{v}, \boldsymbol{u})$ in the stream as graph is undirected. Edges can appear in any order. We can find the detail code in[ADRM].

Where the expected number of iterations is $O(log\, n)$ and we can see each iteration requires a constant number of passes hence total number of passes are $O(log\, n)$

Implementation and Results

The important point here is that above algorithm was implemented using external sort.



In above graph total number of vertices are 10^3. We have tested the algorithm for number of edges starting from linear,$O(n)$, to maximum edges possible that is $\binom{n}{2}$ or $O(n^2)$. Here it can be noticed easily that in later case we can tell, in 1 or 2 iterations, connected components of the graph.

Minimum Spanning Tree in Stream-Sort Model

We use the above algorithm as a subroutine to find MST in stream-sort model. We will see that it has much better results than **semi-streaming model**. The algorithm used here is divide and conquer. The algorithm is as follows

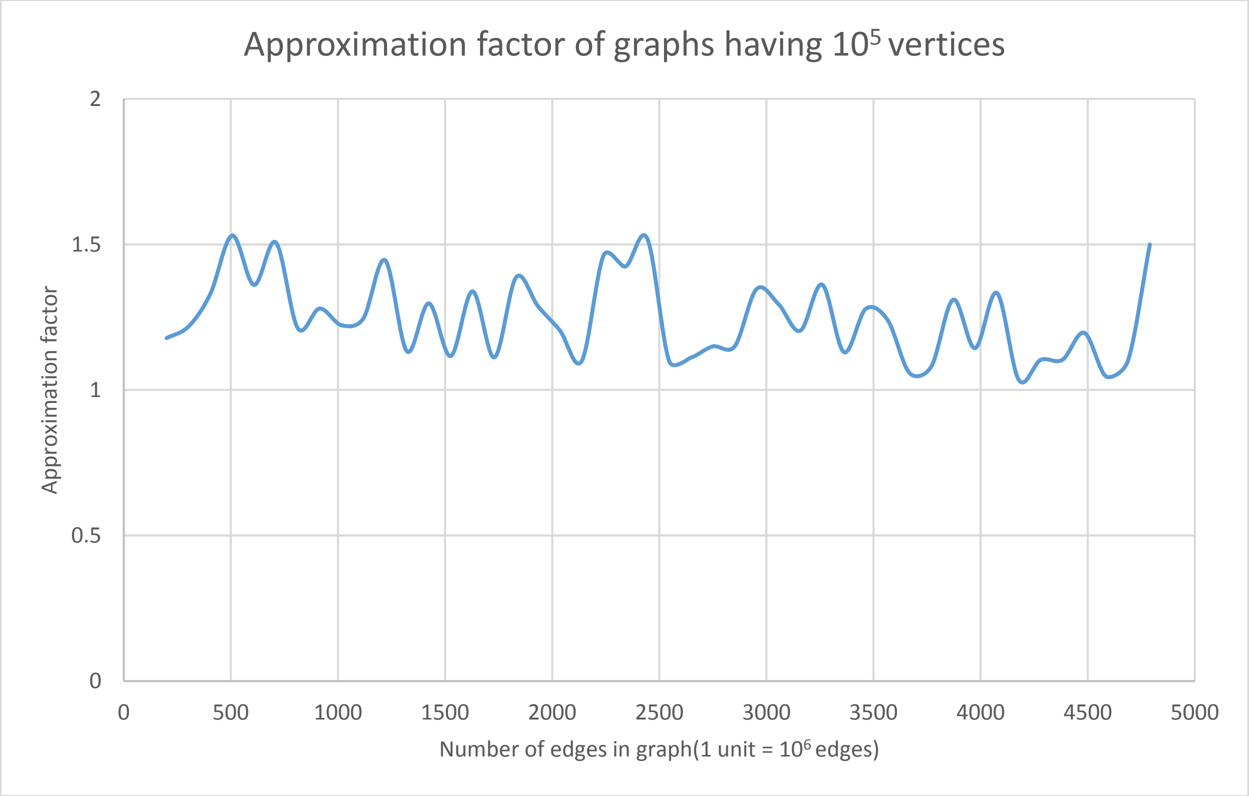
The Algorithm
MST(*V,E*)

```
Sort the edges by increasing weights.
$E_h \longleftarrow$ the lighter half edges.
Compute the connected components of $(V, E_h)$.
$weight \longleftarrow 0$.
for each connected components of $(V, E_h)$, do
    Compute MST recursively and add their weights.
    ($weight \longleftarrow weight + weight\_thisMST$ )
Now treat each components obtained in previous steps as a node of graph and find the edges connecting these nodes.
Compute the MST of this new graph and add its weight.
($weight \longleftarrow weight + weight\_thisMST$)
return weight
```

Having one edge is the bottom case for the above recursive algorithm. Use of sorting the edges in this algorithm is similar to the Kruskal's algorithm but it is done in a way that it is compatible with streaming computations.

Implementation and Results

Analysis of approximation was



As the algorithm is using sorting, so it was expected to give better bound than **semi-streaming**.

We used random graphs having 10^5 vertices. We can see an interesting fact that approximation factor is now not depending on the number of edge. It is fiving almost same results for large number of edges. So we can see it as a very good streaming version of Kruskal's algorithm.

Chapter 4

An Approximate Algorithm to Find Minimum Enclosing Ball in Higher Dimensions

Now we present another Minimum Enclosing Ball Algorithm [AS]. This algorithm has a lower approximation factor than the previous one. It use the concept of corsets and Blurr Ball

Blurred Ball

This section defines the notion of blurred ball cover and describes an algorithm for maintaining such a cover in. For a parameter $0 <\epsilon \leq 1$, an ϵ -blurred ball cover of a set S of n points in R^d, denoted by $K = K(S)$, is a sequence $<K_1, K2, . . . , Ku>$, where each $K_i \subseteq S$ is a subset of $O(1/\epsilon)$ points that satisfies the following three properties;

let B_i = MEB(Ki) and K = unioun(all K_i).

P1 $r(B_{i+1}) \geq (1 + \epsilon^2/8)r(B_i)$ $\forall 1 \leq i < u$

P2 $K_i \subset (1+\epsilon)B_j$ $\forall$ i < j

P3 $\forall$ $p \in S$, $\exists i \leq u$ such that $p \in (1 + \epsilon)B_i$.

Algorithm describes a simple procedure called **Update(K,A)**, that given $K := K(S)$ and a set of $A \subset R^d$, computes $K(S \cup A)$. If we update K as each new point arrives, then A consists of a single point. However, as we will see below, it will be more efficient for some of our applications to update A in a batched mode. That is, newly arrived points are stored in a buffer A and when its size exceeds certain threshold, **Update** procedure is called to update K. Update relies on a procedure **Approx-MEB(Z, ϵ)** that takes a set Z of points and a parameter $0 < \epsilon \leq 1$ and returns a set $G \subseteq Z$ of $O(1/\epsilon)$ points and $B = MEB(G)$ such that $Z \subset (1 + \epsilon)B$.[MBKL].

Update procedure

If there is a point in A that does not lie in the union of the ϵ expansions of B_i ‘s then the **Update** procedure invokes **Approx-MEB** on $K \cup A$ with approximation ratio $\epsilon/3$. Let $K^* \subseteq K \cup A$ be the point set and $B^* = MEB(K^*)$ be the ball returned by Approx-MEB. The Update procedure adds K^* to K and then deletes all K_i’s for which $r(Bi) \leq$"$r(B*)/4$.

The Algorithm [AS]

Update(K,A)

```
if ∃p ∈ A, ∀i ≤ u, p does not belong to (1 +ϵ)Bi then
        K∗,B∗ := Approx-MEB((K ∪A), ϵ/3).
        K_D := {K_i | r(B_i) ≤ ϵr(B∗)/4}
        K := (K \ K_D) o <K∗>
end if
```

Lemma 1

> Let P be a set of points in Rd and let B = MEB(P). Then any closed half space that contains c(B) also contains at least one point of P that is on δB. [AS]

Lemma 2

> For any i < u, $r(B_i+1) \geq (1+\epsilon^2/8)r(B_i)$.[AS]

Lemma 3

> For i < j, $c(B_i) \in (1 + \epsilon)B_j$ [AS]

Lemma 4

> For all $K_i \in K_D$, $(1 + \epsilon)Bi \subseteq (1 + \epsilon)B*$ $[AS]$

Size (Space complexity)

Let $r_i = r(B_i)$. Update ensures that $r_u/r_1 \leq 4/\epsilon$. By (P1), $r_i+1 \geq (1+\epsilon^2/8)r_i$.

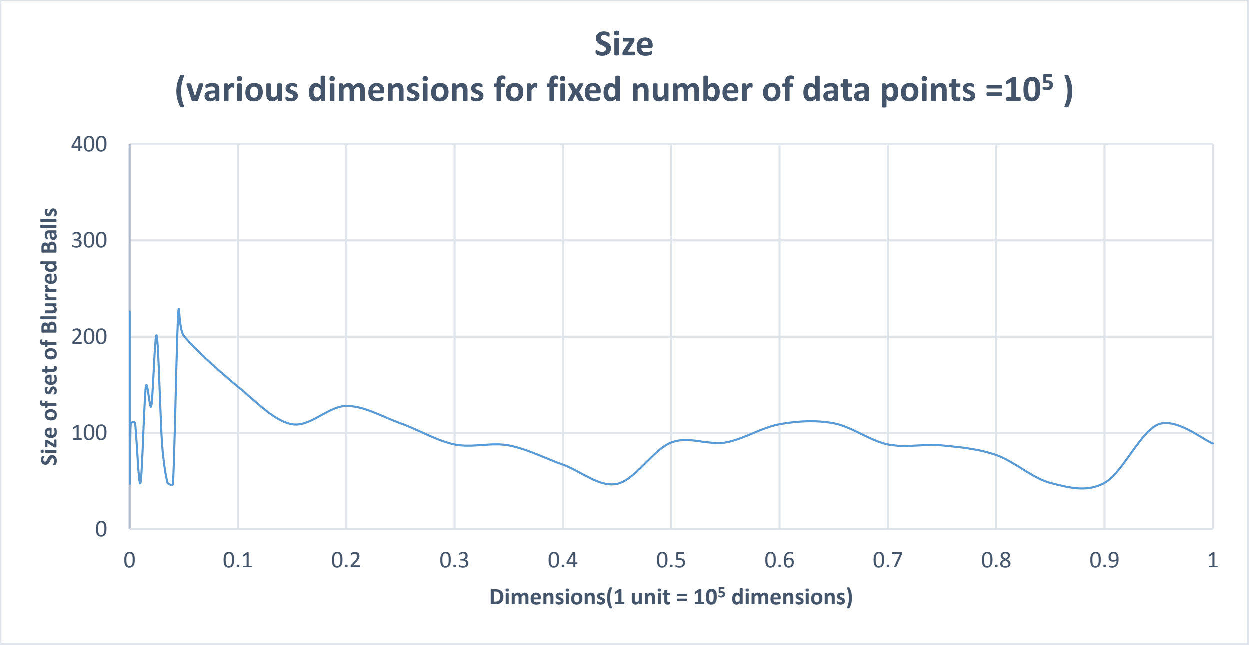
Therefore $u \leq \lceil\log_{1+\epsilon 2/8}(4/\epsilon)\rceil = O((1/\epsilon^2) \log(1/\epsilon))$.

Hence $|K| \leq \Sigma|K_i| = O((1/\epsilon^3) \log(1/\epsilon))$.

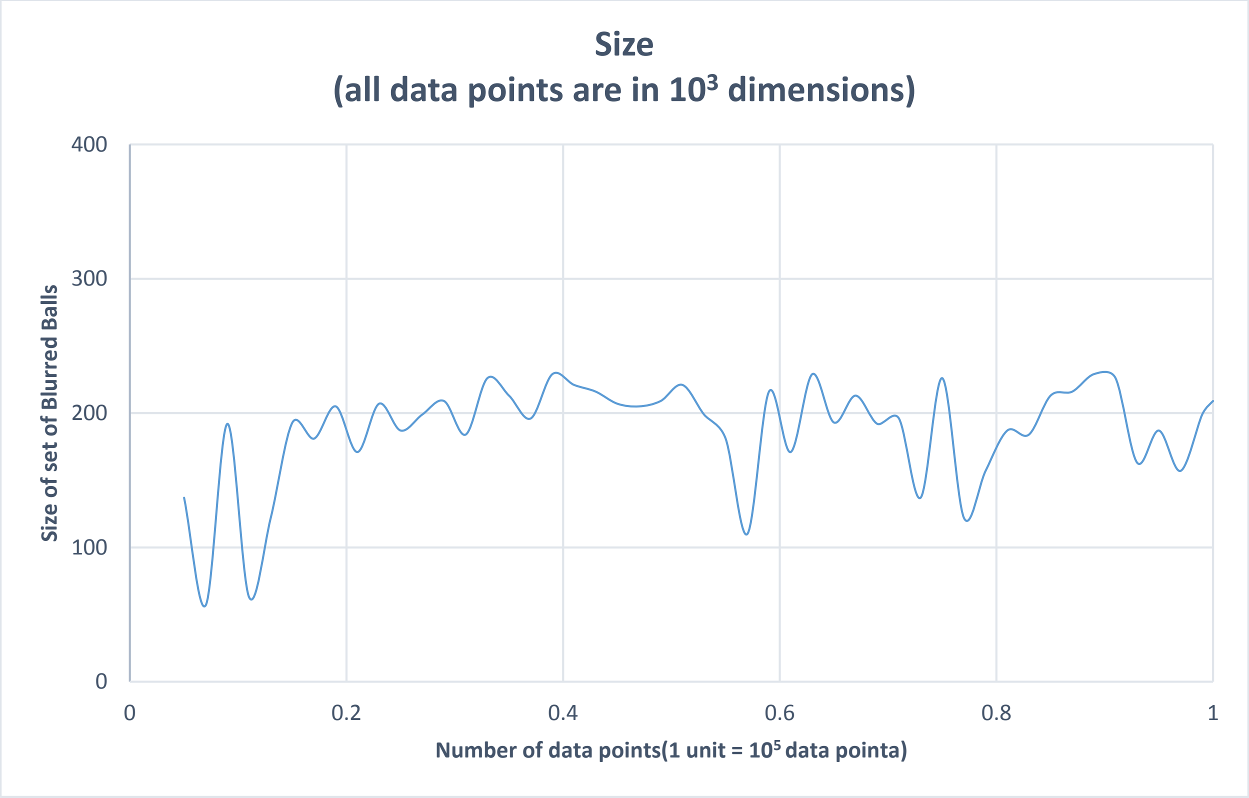
Implementation and Result

$\epsilon = 0.1$ $\qquad ((1/\epsilon)^2 \log_2(1/\epsilon) = 332.192$

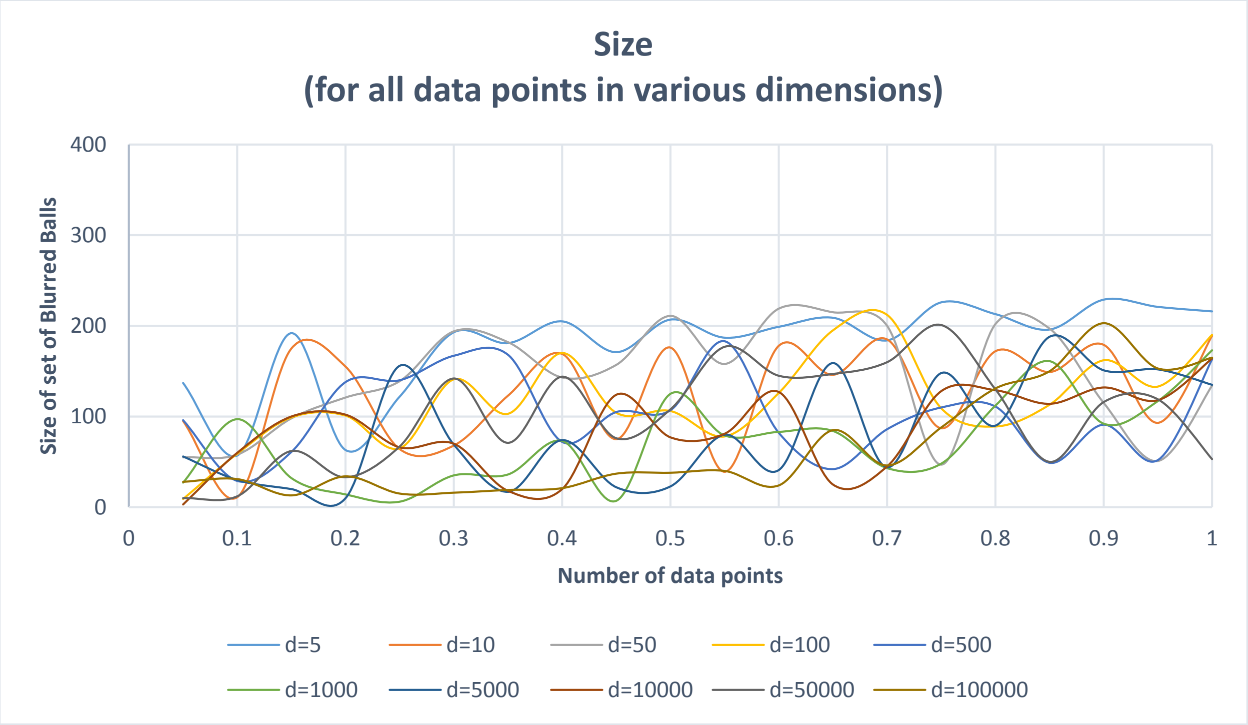
Effect of dimensions



Effect of number of data points



Effect of data points and dimensions



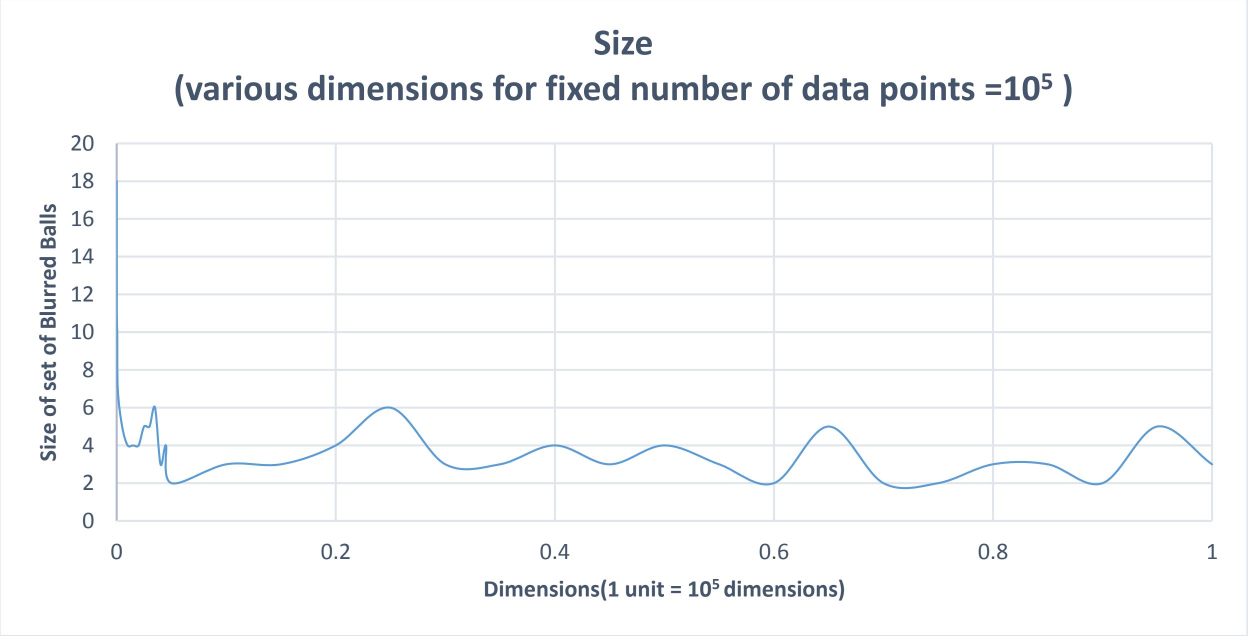
We can see that

At large no of data points in low dimensions, we get bigger size.

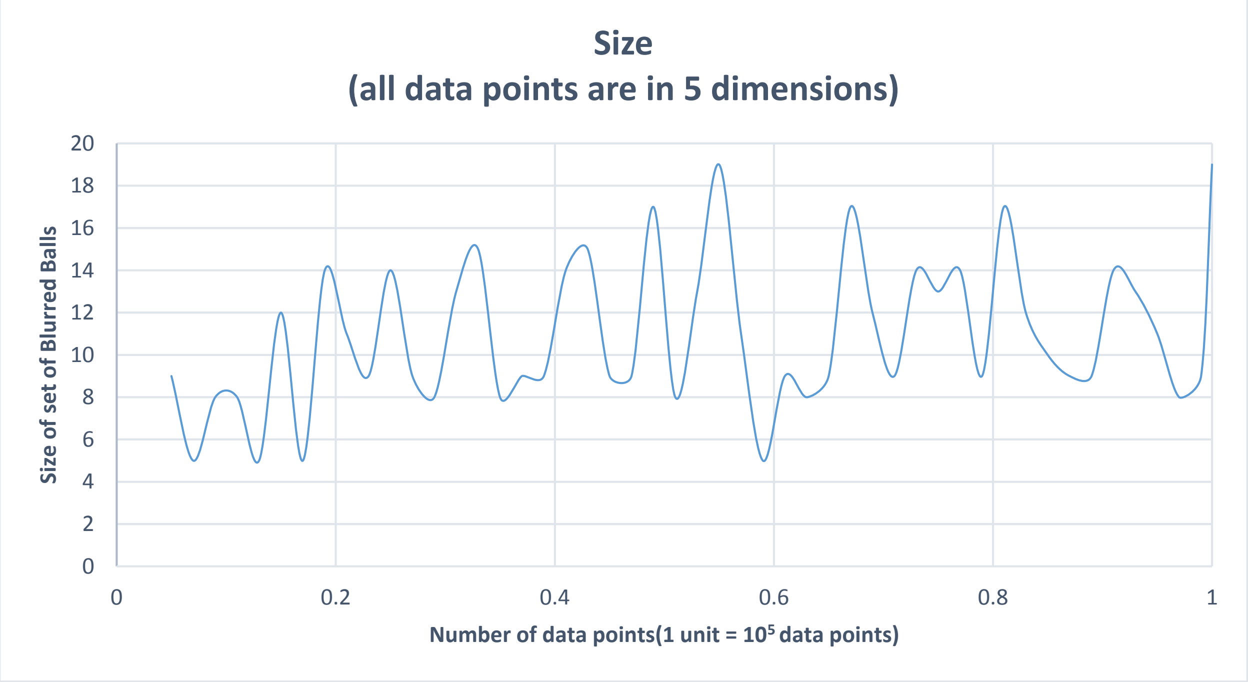
Similar observations are seen in results for other values of epsilon

$\epsilon = 0.3$ $((1/\epsilon)^2 \log_2(1/\epsilon) = 19.2996$

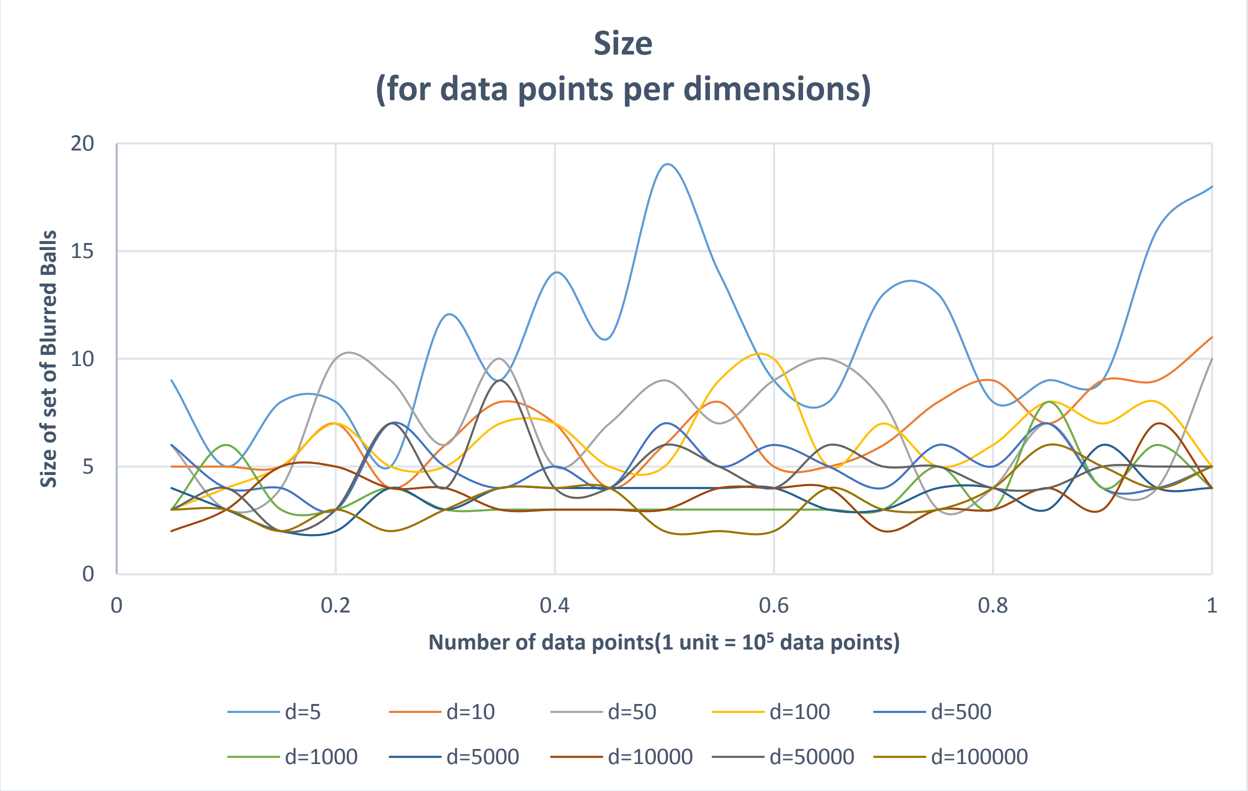
<u>Effect of dimensions</u>



Effect of number of data points



Effect of data points and dimensions



So we can observe that

Size is bigger for large no of data points.

Size is bigger for smaller dimensions.

Applications of Blurred Ball Cover in Minimum Enclosing Ball

For a stream S of points, we maintain a (ϵ/9)-blurred ball cover K of S.

K is updated whenever a new point arrives. Let B = {B_1, . . . ,B_u}. Let B∗ = MEB(B). We return (1+ϵ/3)B∗ which can be computed in time O(1/ϵ^5) [KMY].Hence the total update time is O(1/ϵ^5).

Let Let r’ = r(MEB(S))

Lemma 5

r(B∗) ≤ ((1 + √3)/2 + ϵ/3)r’[MS]

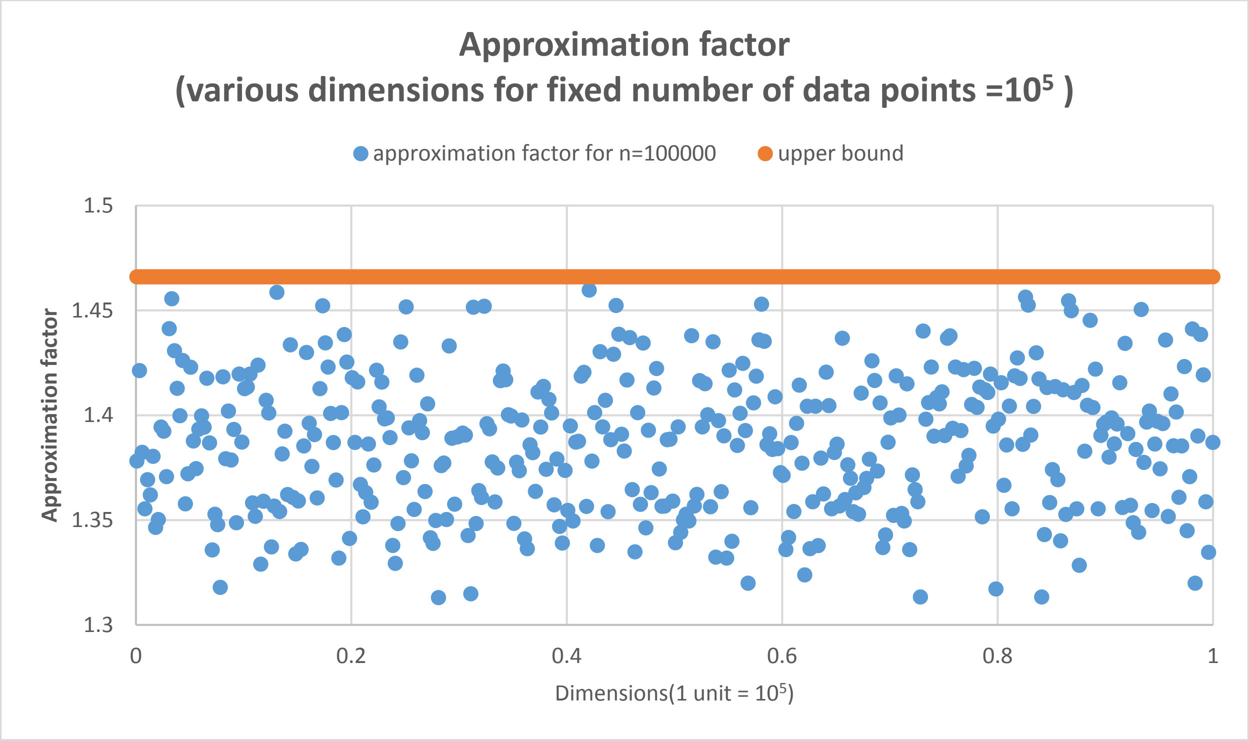
This lemma implying that (1 + ϵ/3)B∗ is indeed a((1 + √3)/2 + ϵ)-MSB and we conclude the following

> **Given a stream S of points in R^d, with the use of size O((d/ϵ3) log(1/ϵ)) that maintains a ((1 + √3)/2 + ϵ)-MSB.** [MS]

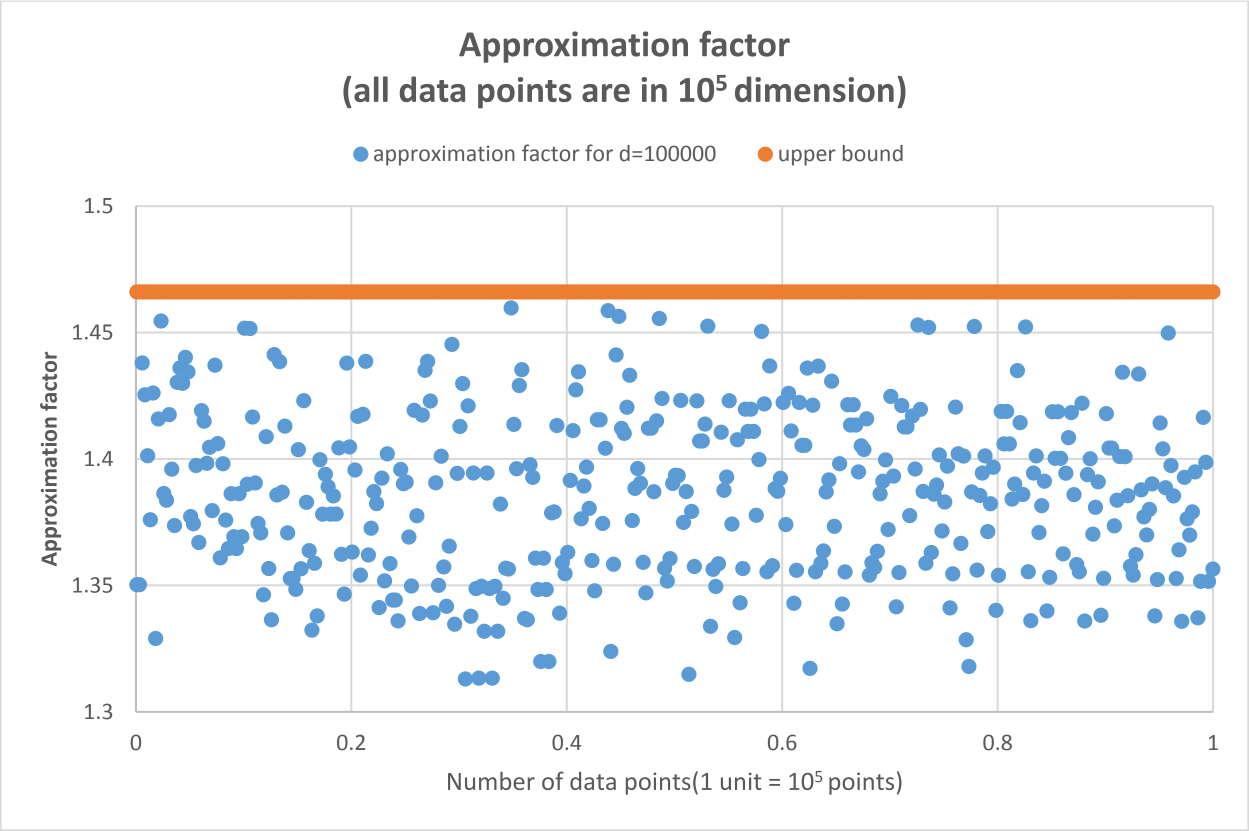
Implementation and Result

$\epsilon = 0.1$ $(1+\sqrt{3})/2 + \epsilon = 1.466$

To check the effect of dimensions on data points, dimensions varies from 10 to 10^5 for fixed 10^5 data points.

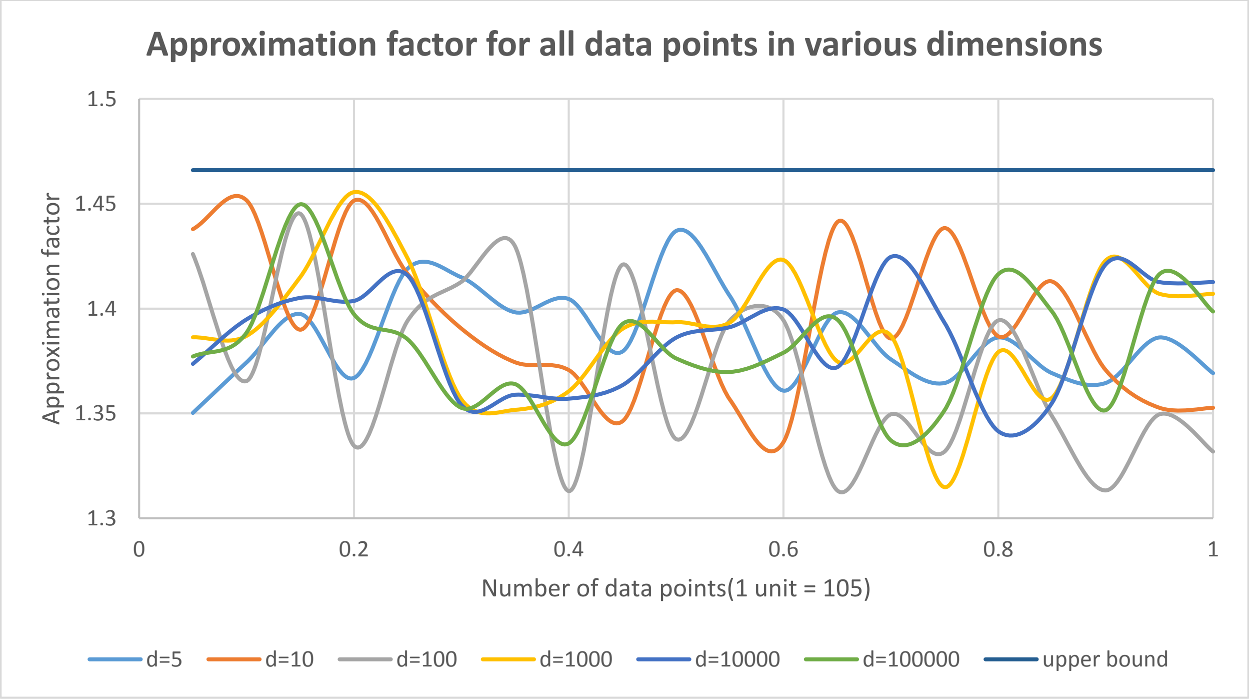


To check the effect of number of data points, no. of data points varies from 10 to 10^5 in fixed 10^5 dimension



To check the effect of data points and dimensions in one plot

(Here all the result-points are not used for plotting to make plot clear)



We can see that

No instance crossed the upper bound.

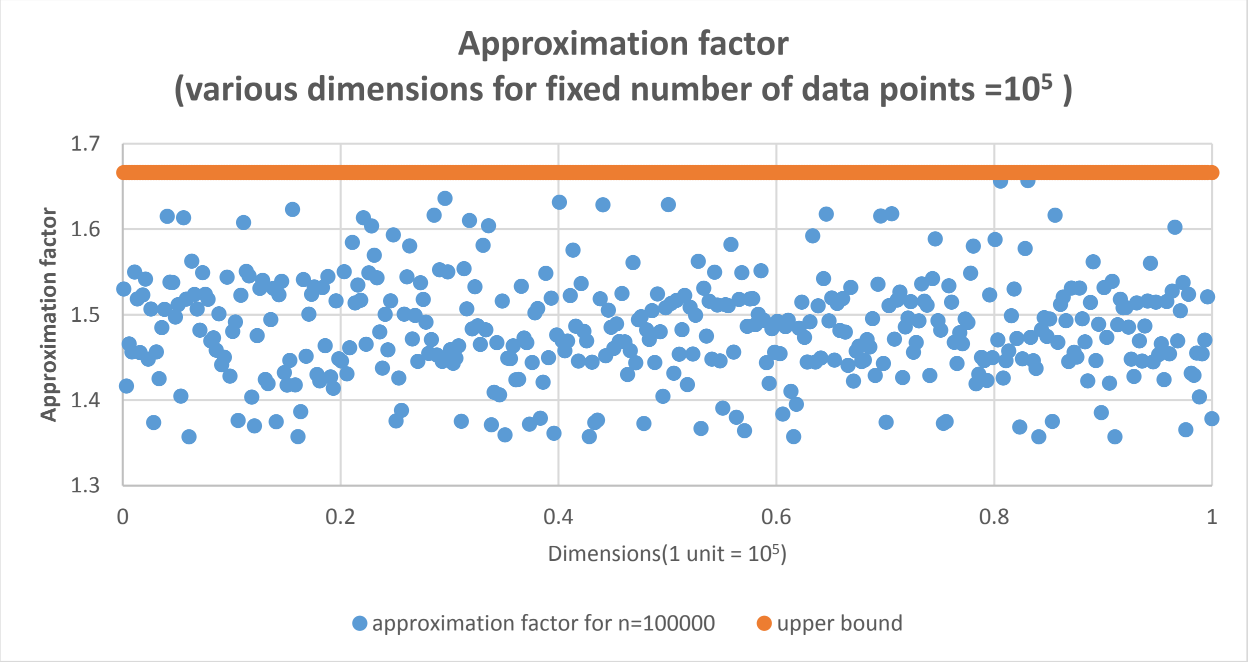
None went below a certain point

Does above algorithm has **some lower bound**, say 1.3 or 1.2? (Not proven/analyzed in the original paper but results, after implementing it, strongly suggest this)

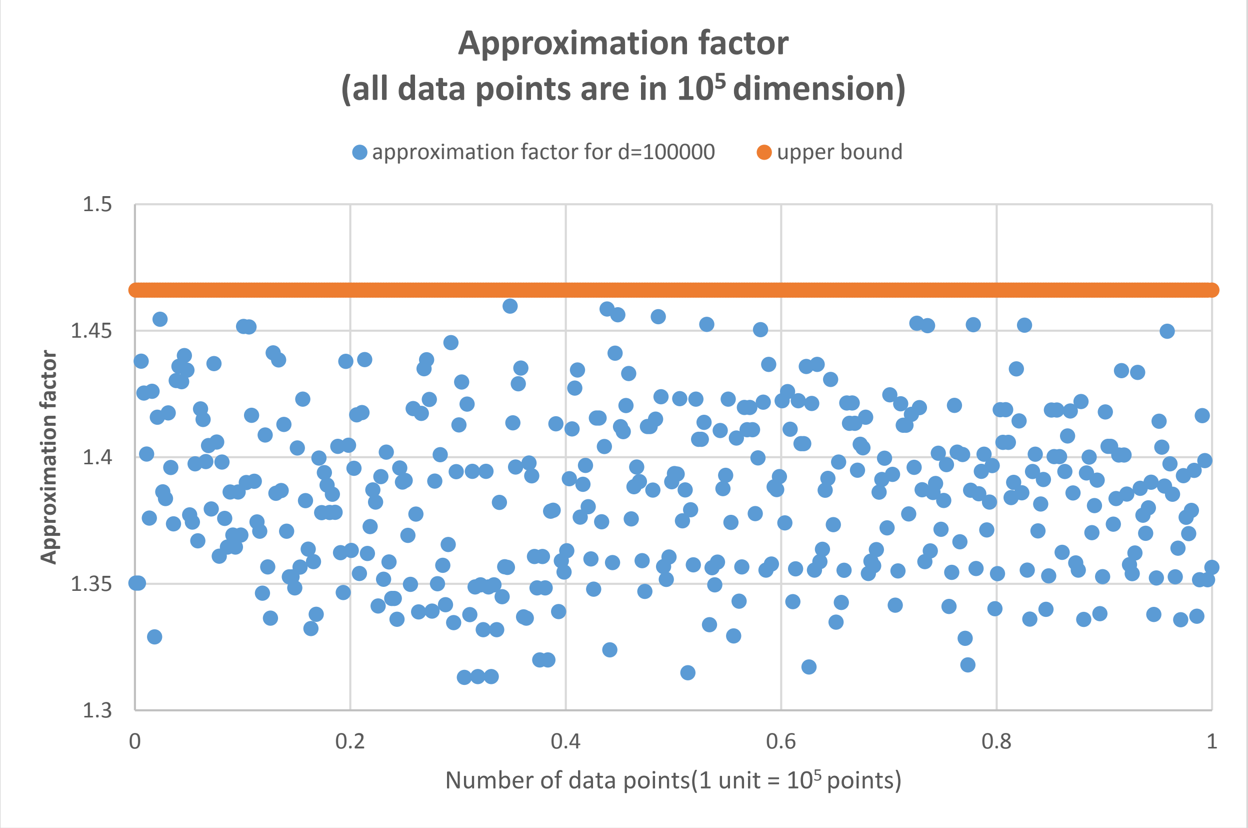
Similar results were found for different values of epsilon

ϵ = 0.3 (1+√3)/2 + ϵ = 1.666

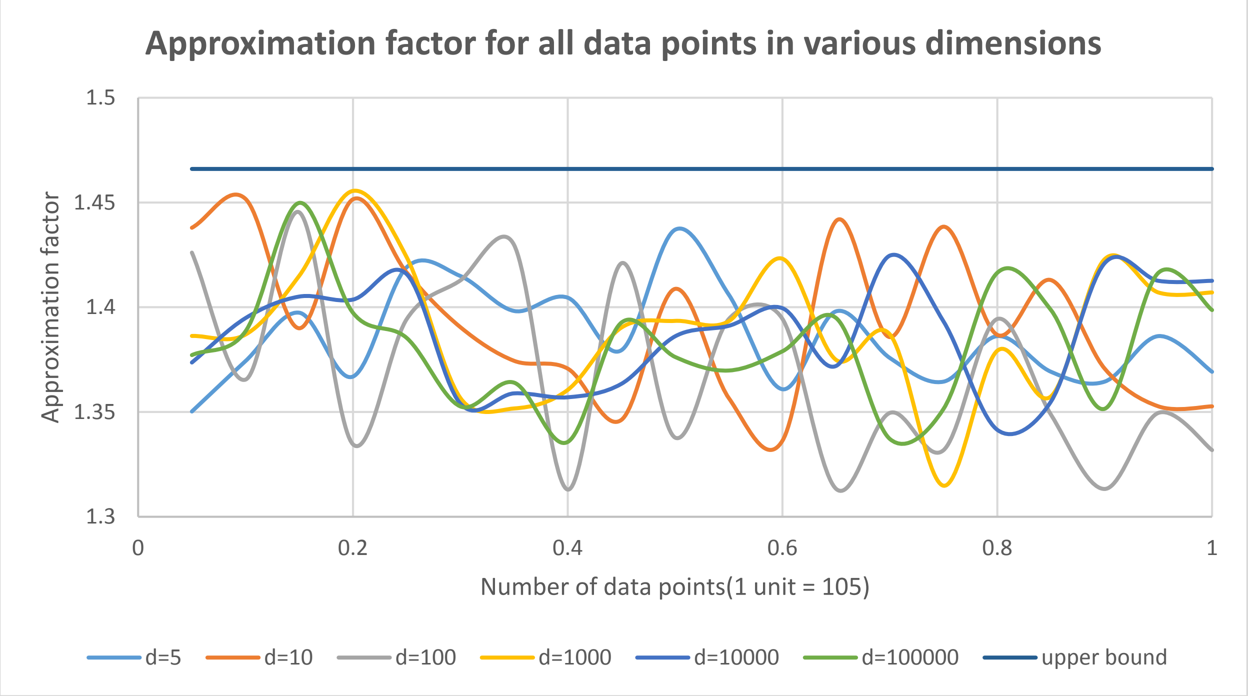
To check the effect of dimensions on data points, dimensions varies from 10 to 10^5 for fixed 10^5 data points



To check the effect of number of data points, no. of data points varies from 10 to 10^5 in fixed 10^5 dimension.



To check the effect of data points and dimensions in one plot



Bibliography

[ADR+04]

Gagan Aggarwal, Mayur Datar, Sridhar Rajagopalan, and Matthias Ruhl. On the streaming model augmented with a sorting primitive. In FOCS ’04: Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS’04), pages 540–549, Washington, DC, USA, 2004. IEEE Computer Society.

[DFR06]

Camil Demetrescu, Irene Finocchi, and Andrea Ribichini. Trading off space for passes in graph streaming problems. In SODA ’06: Proceedings of the seventeenth annual ACMSIAM symposium on Discrete algorithm, pages 714–723, New York, NY, USA, 2006. ACM Press

[FKM+05b]

Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian

Zhang. Graph distances in the streaming model: the value of space. In SODA ’05:

Proceedings of the sixteenth annual ACM-SIAM symposium on discrete algorithms, pages

745–754, Philadelphia, PA, USA, 2005. Society for Industrial and Applied Mathematics

[FKM+05a]

Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian

Zhang. On graph problems in a semi-streaming model. Theoretical Computer Science,

348(2-3):207–216, 2005

[R03]

Matthias Ruhl. Efficient Algorithms for New Computational Models. Ph.D. Thesis, MIT,

Cambridge, MA, USA, 2001

[T83]

Robert Endre Tarjan. Data structures and network algorithms. Society for Industrial and

Applied Mathematics, Philadelphia, PA, USA, 1983.

[V01]

Jeffrey Scott Vitter. External memory algorithms and data structures: dealing with massive data. ACM Comput. Surv., 33(2):209–271, 2001

[MBKL]

M. Bˇadoiu and K. L. Clarkson, Optimal core-sets for balls, *Comput. Geom. Theory Appl.*, 40 (2008), 14–22

[KMY]

P. Kumar, J. S. B. Mitchell, and E. A. Yildirim, Approximate

minimum enclosing balls in high dimensions

using core-sets, *J. Exp. Algorithmics*, 8 (2003), 1.1

[AS]

Streaming algorithms for extent problems in higher dimensions. Pankaj Agrawal, R.SharatKumar

[ADRM]

On the Streaming Model Augmented with a Sorting Primitive

Gagan Aggarwal, Mayur Datar Sridhar Rajagopalan Matthias Ruhl