Analysis and Implementations of
Streaming Algorithms in Computational Geometry

A

Bachelor Thesis Project

Report Submitted in Partial Fulfillment of the

Requirements for the Degree of

Bachelor of Technology

by

Shrinivas Acharya

(10010164)

Under the guidance of

Dr. S.V. Rao

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI

GUWAHATI - 781039, ASSAM

CERTIFICATE

*This is to certify that the work contained in this thesis entitled "**Analysis and Implementations of Streaming Algorithms in Computational Geometry**" is a bona fide work of **Shrinivas Acharya (Roll No. 10010164)**, carried out in the Department of Computer Science and Engineering, Indian Institute of Technology Guwahati under my supervision and that it has not been submitted elsewhere for a degree*.

Supervisor: **Dr. S. V. Rao**
Associate Professor,
Department of Computer Science & Engineering,
Indian Institute of Technology Guwahati, Assam.

April,2014
Guwahati.

Acknowledgements

I acknowledge with gratitude the constant encouragement and inspiration provided by my guide Dr. S. V. Rao and thank him for his able guidance, keen interest and valuable suggestions in the preparation of this thesis. Under his guidance I was not only able to appreciate a new domain of research but also realized the essential elements of research like defining a problem, constructing an approach, taking relevant assumptions, among others.

My thanks are also due to my friends and a number of people, whom I discussed with the various nuances of the problem at hand that helped me give a better structure to my ideas.

Last, but surely not the least, I am deeply grateful to my family, for the endearing encouragement and emotional support.

Shrinivas Acharya

Abstract

The problem of finding the minimum spanning tree (MST) has several applications specifically in design of networks, including computer networks, water supply networks, electric grids etc. But in real life scenarios the size of graph is too large to fit inside the main memory so problem of finding the MST in streaming computations.

Finding the minimum enclosing ball (MEB) problem is presented in this report. The MEB problem has various important applications, which often requires it be solved in relatively high dimensions. Some of the applications of MEB computation include gap tolerant classifiers [8] in Machine Learning, tuning Support Vector Machine parameters [10], Support Vector Clustering [4; 3], preprocessing for fast farthest neighbor query approximation [19], k-center clustering [5], testing of clustering [2], solving the approximate 1-cylinder problem [5], computation of spatial hierarchies (e.g., sphere trees [20]), and other applications [13].

Chapter1

Introduction

Steaming algorithms have been studied in various computation models, three of them are Semi-Streaming, W-Streaming and Stream-Sort model. Each model is defined on some criteria such as of number of passes required or size of intermediate streams generated etc. Whether the minimum spanning tree (MST) can be computed or not, in each model, will be discussed with these models.

The implementations and results are presented for each model.

Finding the minimum enclosing ball (MEB) problem is presented in this report. The MEB problem has various important applications, which often requires it be solved in relatively high dimensions. Some of the applications of MEB computation include gap tolerant classifiers [8] in Machine Learning, tuning Support Vector Machine parameters [10], Support Vector Clustering [4; 3], preprocessing for fast farthest neighbor query approximation [19], k-center clustering [5], testing of clustering [2], solving the approximate 1-cylinder problem [5], computation of spatial hierarchies (e.g., sphere trees [20]), and other applications [13].

One special technique of core-sets is used here to compute the MSB which is highly used in data clustering problems. The implementation and results for this algorithm is presented here.

Chapter2

Finding Minimum Spanning Tree in Various Models of Streaming Algorithms

Number of passes required is the key role in any streaming algorithm. But some models also allow to generate intermediate streams. So based on theses, there are three main models of streaming algorithms:

1. Semi-Stream
2. W-stream
3. Stream-Sort

As we are presenting the MST algorithms let us introduce few notations.

Notations

1. $\boldsymbol{n} =$ Number of vertices in graph
2. $\boldsymbol{e} =$ Number of edges in graph
3. $polylog\ n = log^k n$, where k is some constant.

2.1 Semi-Stream

In this model, the algorithm is given $\Theta(n\ polylog\ n)$ space where n is defined earlier.

So this is a model where the algorithm has enough internal memory to store the vertices but not necessarily the edges in the graph.

2.1.1 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 MST. The complete algorithm is as follows:

2.1.2 Algorithm [FKM+05b]

```
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.
```

2.1.3 Analysis

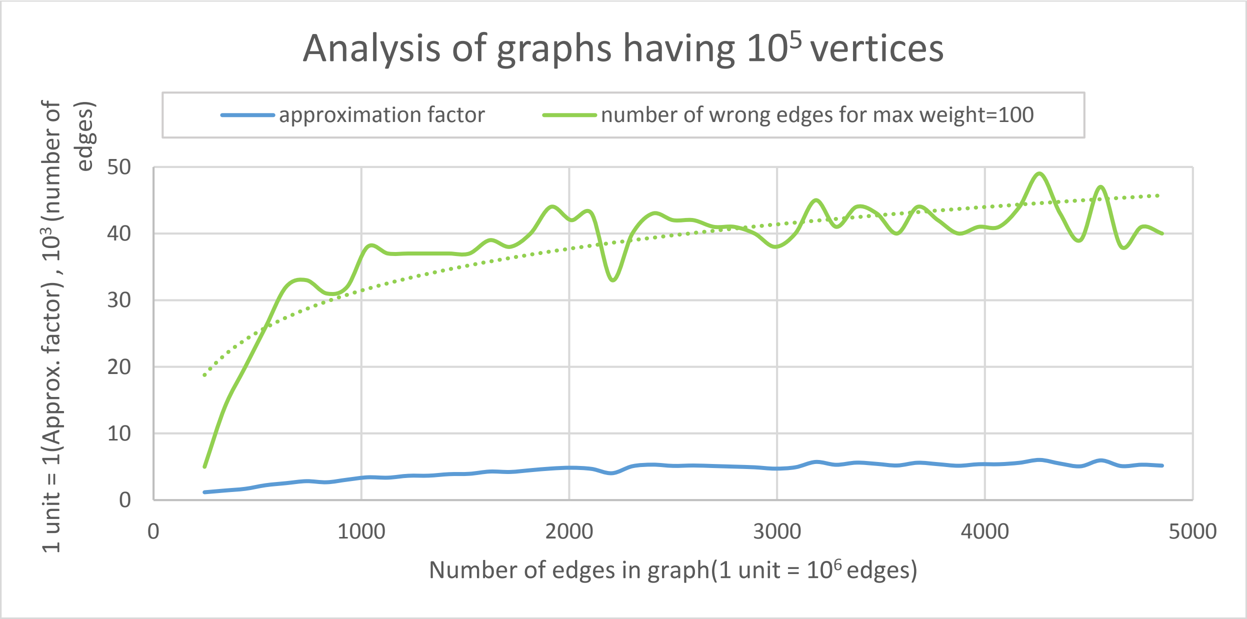
We are storing only edges of current spanning tree in each step or iteration. As the number of edges in spanning tree are equal to $O(n)$, so it takes $O(n)$ space (which is equals to $O(n\ ploylog\ n)$space) that meets the criteria for semi-streaming model.

2.1.4 Implementations and Results

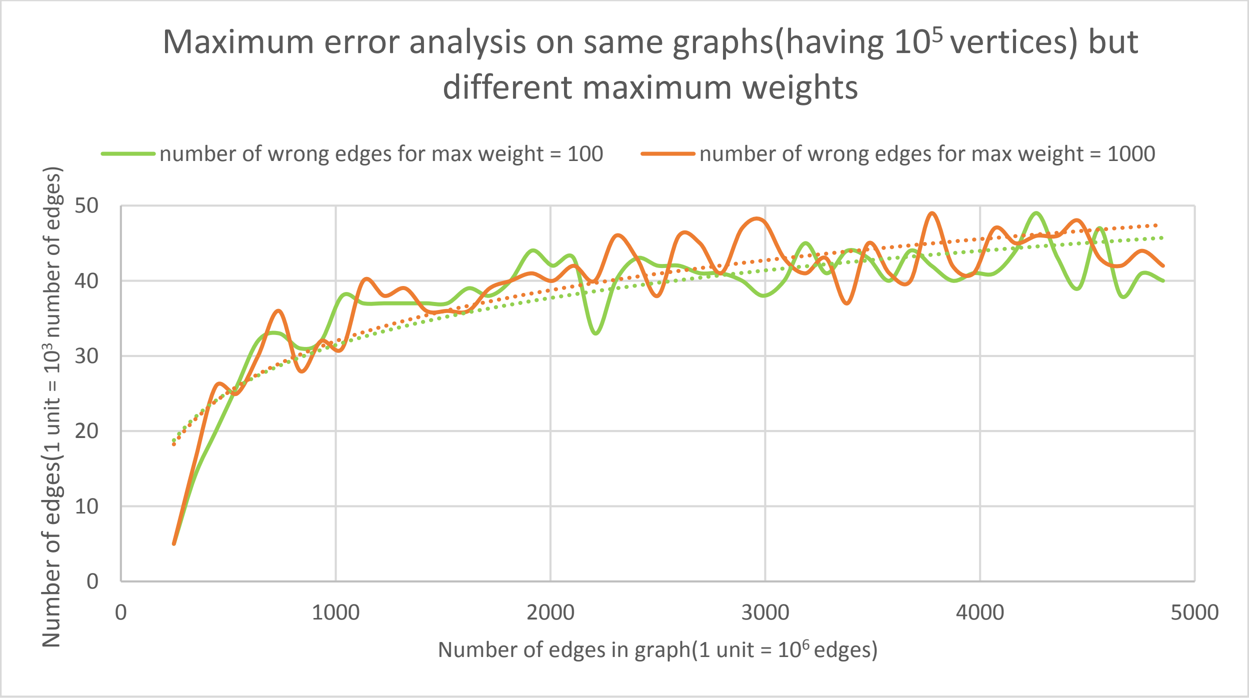
Above algorithm was efficiently implemented in C and results are obtained.

Analysis of approximation factor is done by comparing its results with Prim's algorithm of finding Minimum spanning tree.

The algorithm is tested for different-different values of maximum weights and difference between maximum and minimum weights.


Analysis of graphs having 10^5 vertices
approximation factor
number of wrong edges for max weight=100
1 unit = 1(Approx. factor) , 10^3 (number of edges)
0
10
20
30
40
50
0
1000
2000
3000
4000
5000
Number of edges in graph(1 unit = 10^6 edges)

Plot.1


Maximum error analysis on same graphs(having 10^5 vertices) but different maximum weights
number of wrong edges for max weight = 100
number of wrong edges for max weight = 1000
Number of edges(1 unit = 10^3 number of edges)
0
10
20
30
40
50
0
1000
2000
3000
4000
5000
Number of edges in graph(1 unit = 10^6 edges)

Plot.2

2.1.4 Observations

As we can see in Plot.1 that approximation factor is increasing with number of edges and reaches as high as 5. But comparing approximation factor is not a good way 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 MST returned by Prim's algorithm is used as optimal MST) and call it error.

So error is not 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 in Plot.1 and when maximum weight is 1000 in Plot.2

2.1.5 Conclusion

Surely this model computes the MST in on pass but it makes significant amount of error.

2.2 W-Stream

For Semi-Stream model, the crucial limitation is that the streaming model is unable to modify the data stream. In this model ([R03]) the algorithms can not only process the stream but also 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.

So in this model algorithm have power to generate intermediate streams (by modifying the input stream) but not more than $O(n)$ size.

2.2.1 Minimum Spanning Tree

A deterministic W-Stream algorithm for finding the Connected Components (CC) of an undirected graph that uses s space and $O((n \, log \, n)/s)$ 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.

2.3 Stream-Sort

W-Stream has much power in comparison to Semi-Stream but still some problems require more. As some of the problems, like finding MST using Connected Components (CC), can't be solved in W-Stream mode hence a model with some extra abilities is emerged.

In this model, in each pass through the data, we can not only produce an intermediate stream (as in the W-Stream) but also 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 $O(polylog\ n)$

2.3.1 Connected Components as Subroutine for MST

MST uses Connected Components as its subroutine, so we present the algorithm to find all the connected components in an undirected graph.

2.3.1.1 **Algorithm** [ADRM]

We assume that our stream is the list $\boldsymbol{V}$ of the vertices $(v_1, v_2, \dots\dots, v_n)$ followed by the list $\boldsymbol{E}$ of the vertex-pairs representing the edges. Since the graphs we study are undirected, $\boldsymbol{E}$ contains both pairs (u, v) and (v, u) for an edge between u and v.

While both $\boldsymbol{V}$ and $\boldsymbol{E}$ are part of one stream, we will sometimes state operations on the two halves independently, assuming that in such a pass, the other half of the stream is passed through unchanged.

1: First, in a pass over $\boldsymbol{V}$, we produce a stream $\boldsymbol{R}$ that contains pairs of node names $\boldsymbol{v_i}$ and random distinct numbers r_i , e.g. 3logn-bit random numbers:

$$(v_1, r_1)(v_2, r_2)(v_3, r_3) \dots\dots (v_n, r_n)$$

2: Now we determine the new label for each node, i.e. the lowest value of r_i among its neighbors and itself.

a. First, we sort the pair of streams $\boldsymbol{R}$ and $\boldsymbol{E}$, so that the pairs (v_i, r_i) are directly followed by all edges whose first component is v_i :

$$(v_1, r_1)(v_1, \_\ )(v_1, \_) \dots\dots (v_2, r_2)(v_2, \_\ )(v_1, \_) \dots\dots$$

In one pass on this stream, we can produce a new stream of edges $\boldsymbol{E'}$, where for the first component of each edge, v_i is replaced by the corresponding r_i :

$$(r_1, \_\ )(r_1, \_) \dots\dots (r_2, \_\ )(r_2, \_) \dots\dots$$

b. Now we sort $\boldsymbol{R}$ and $\boldsymbol{E'}$, so that the pair (v_i, r_i) sappear right before all edges whose second component is equal to v_i :

$$(v_1, r_1)(\_, v_1)(\_, v_1) \ldots \ldots (v_2, r_2)(\_, v_2) \ldots \ldots$$

c. In this stream, the number r_i assigned to a node v_i occurs right before the list of numbers assigned to the nodes incident to v_i. Thus, in one linear pass, we can determine the smallest number among them for each v_i, which yields a stream $\boldsymbol{L}$ of nodes v_i with their new labels l_i:

$$(v_1, l_1)(v_2, l_2)(v_3, l_3) \ldots \ldots (v_n, l_n)$$

3: Now check if $\boldsymbol{R} = \boldsymbol{L}$ then declare it as **Final Label** else

If not equal then $\boldsymbol{R} \leftarrow \boldsymbol{L}$ and repeat this until we find stream $\boldsymbol{R}$ and $\boldsymbol{L}$ equal.

4: Final label of any vertex is its id in connected components. That means all vertices with same labels belong to same connected components.

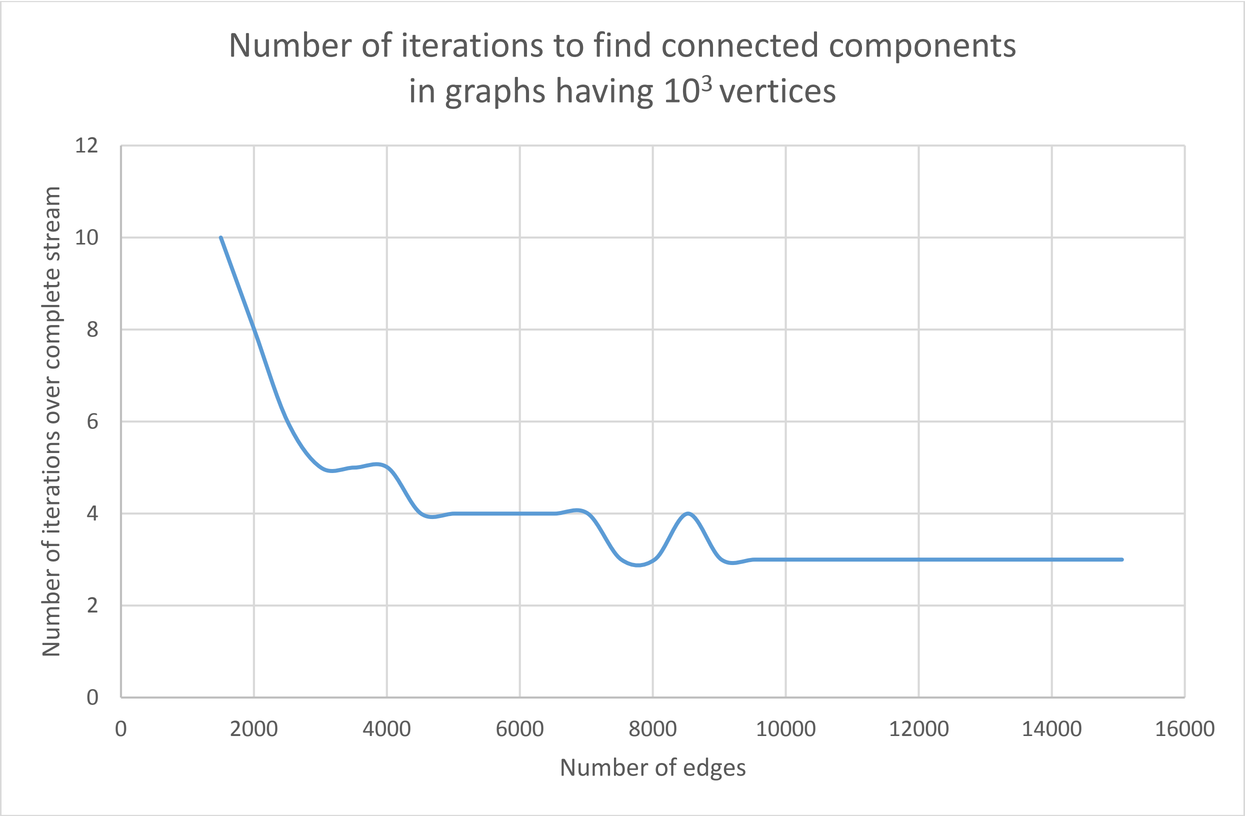
2.3.1.2 Analysis

By assigning random labels to each vertex and merging vertices, the number of vertices in the graph decreases during each iteration and expected number of iteration comes out to be $O(log\ n)$ ([ADR+04]). Use of random numbers make expected number of iteration as $O(log\ n)$.

Every intermediate stream generated is of $\mathrm{O}(stream - size)$, as these are just rearrangement of input streams. So criteria for Stream-Sort is satisfied.

2.3.1.3 Implementations and Results

Above algorithm was separately tested to verify the bounds. External sorting is used to sort the data streams and intermediate streams.



Plot.3

2.3.1.4 Observations

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

In above plot graphs having linear edges in term of number of vertices, are not shown as that would require liner time.

2.3.1.5 Conclusion

Planner graph takes $O(n)$ iterations and dense graph (nearly complete) takes $O(1)$ iterations. But on an average a graph which is not the above two takes $O(\log n)$ iterations.

2.3.2 Minimum Spanning Tree

Finding MST use the above connected components algorithm as its subroutine. The algorithm used here is divide and conquer. The algorithm is as follow

2.3.1.1 Algorithm [ADRM]

We are going to use a divide and conquer approach to compute a minimum spanning tree of a graph$\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$.

```
Sort the edges by increasing weights.
$\boldsymbol{E_h} \longleftarrow$ the lighter half edges.
Compute the connected components of $(\boldsymbol{V}, \boldsymbol{E_h})$ using above algorithm.
$weight \longleftarrow 0.$
**for** each connected components of $(\boldsymbol{V}, \boldsymbol{E_h})$, **do**
    Compute MST recursively (call it $thisMST$) 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(call it* $newMST$) and add its weight.
$weight \longleftarrow weight + weight\_newMST)$
return $weight.$
```

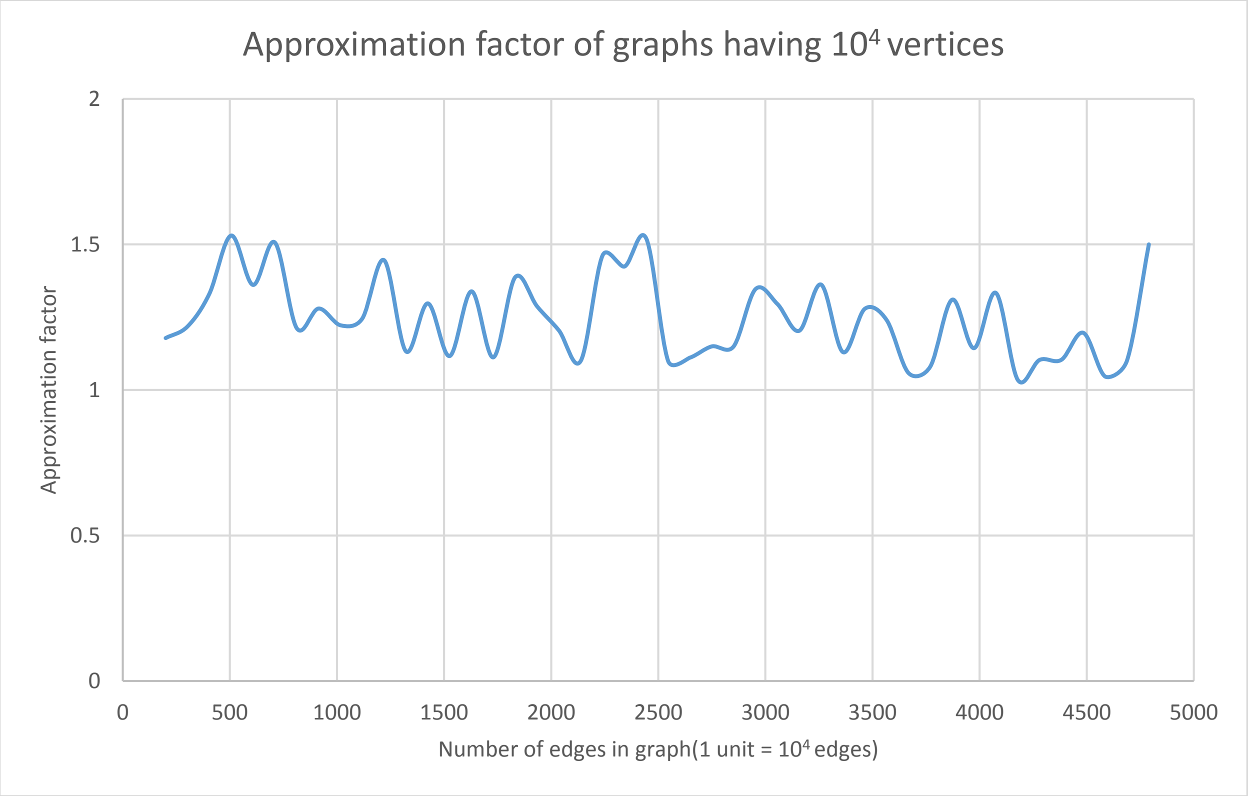
2.3.2.2 Analysis

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

2.3.2.3 Implementations and Results

Above algorithm was efficiently implemented in C and results are obtained.

Analysis of approximation factor is done by comparing its results with Prim's algorithm of finding Minimum spanning tree.



Plot.4

2.3.1.4 Observations

As the algorithm is using sorting, so it was expected to give better bound than Semi-Streaming.

We used random graphs having 10^4 vertices. We observed an interesting fact that approximation factor is now not depending on the number of edge. It is giving almost same results for large number of edges. In Plot.4 we can see that for planner graphs and for dense graphs approximation factor is same and is very good in comparison of Semi-Stream.

So we can see it as a very good streaming version of Kruskal's algorithm.

2.3.1.5 Conclusion

It as a very good streaming version of Kruskal's algorithm which gives approximation factor that does not vary much with number of edges.

Chapter 3

An approximate Algorithm to Find Minimum Enclosing Ball in Higher Dimensions

2.1 Prior Work

An approximate streaming algorithm for computing the minimum enclosing ball (MEB) of n points was implemented previously.
This algorithm has an approximation factor 1.5
Algorithm is briefly described as follows:

The Algorithm[14]

```
Ball $B$
$B.c \longleftarrow p_o$
$B.r \longleftarrow 0$
**For** each point $p$ in the input stream $P$ **do**
    **If** $p$ is outside $B$ **do**
        $r_{i-1} \longleftarrow B.r$
        $c_{i-1} \longleftarrow B.c$
        $\delta_i \longleftarrow \frac{1}{2}(\| p - c_{i-1} \| - r_{i-1})$
        $B.c \longleftarrow c_{i-1} + \frac{\delta_i}{\|p - c_{i-1}\|}(p - c_{i-1})$
        $B.r \longleftarrow r_{i-1} + \delta_i$
```

The above algorithm returns MEB which has radius maximum 1.5 times greater than optimal MEB

This algorithm was implemented and results were tested in previous phase.

2.2 ϵ-Expansion

For a point $x \in R^d$ and a value $r > 0$, let $B(x, r)$ denote a ball of radius r centered at x. For a ball B, let c(B), r(B) denote its center and radius respectively, and let $(1+\epsilon)B$ denote the ϵ-expansion of B.

2.2 Blurred Ball Cover

We define a new term Blurred Ball Cover, which will be key point of our algorithm, as follows :

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 $<K_1, K2, \ldots, Ku>$, where each $K_i \subseteq S$ is a subset of $O(1/\epsilon)$ points that satisfies the following three properties;

Let B_i = MEB(K_i) and $K = \bigcup_{i \leq u} K_i$

1. $r(B_{i+1}) \geq (1 + \epsilon^2/8)r(B_i) \quad \forall 1 \leq i < u$
2. $K_i \subset (1+\epsilon)B_j \quad \forall\, i < j$
3. $\forall\ p \in S, \exists i \leq u$ such that $p \in (1 + \epsilon)B_i$.

2.3 Some Approximate MEB algorithm used as Subroutine

We present here an algorithm to find core-set which will be used in later part of our main algorithm. Problems of finding core-sets are broadly used in data- clustering algorithms. Let us now define what a core-set is.

2.3.1 Core-Set

Given a set of points $P \subset R^d$ and value $\epsilon > 0$, a *core-set S* is a set of points $S \subset P$ such that it has the property that the smallest ball containing S is within ϵ of the smallest ball containing P. That is, if the smallest ball containing S is expanded by $1 + \epsilon$, then the expanded ball contains P.

2.3.2 Algorithm to Find Core-Set

We will use the algorithm to find the core-set given by Mihai Badoiu and Kenneth L. Clarkson [MBKL]. We will directly use this algorithm and its results in our algorithm

Their algorithm takes Z (set of points) and a parameter ϵ as input and outputs a set of size $O(1/\epsilon)$. Let call this Approx-MEB(Z,ϵ). The algorithm is available at [MBKL]. It is implemented in C to make compatible with our final algorithm.

2.4 Algorithm to maintain Blurred Ball Cover

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, that means, 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(K,A) 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 $Z \subset (1 + \epsilon)B$. It is described in previous section.

2.4.1 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 $r(B_i) \leq \epsilon r(B^*)/4$.

Update(K,A) [AS]

```
if $\exists p \in A, \forall i \leq u$, p does not belong to $(1 +\epsilon)Bi$ then
        $K^*, B^* := Approx\text{-}MEB((K \cup A), \epsilon/3)$.
        $K_D := \{K_i \mid r(B_i) \leq \epsilon r(B^*)/4\}$
        $K := (K \setminus K_D) \circ \langle K^* \rangle$
end if
```

2.4.1.1 Statement and its Proof of Correctness

Statement: All three properties of blurred-ball cover are maintained during update procedure.

Now we will prove this statement

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]

This is the property that every MEB satisfies.

Lemma 2: For all $i < u$, $r(B_{i+1}) \geq (1+\epsilon^2/8)r(B_i)$.

Proof is as follows:

Let $B' = MEB(K \cup \{q\})$.

Clearly $r(B_{i+1}) \geq r(B')$ { as B′ is MEB which has }

Let $r' = r(B')$, $c' = c(B')$, $c_i = c(B_i)$, and $r_i = r(B_i)$.

We will prove that $r' > (1+\epsilon^2/8)r_i$. Which will prove our Lemma.

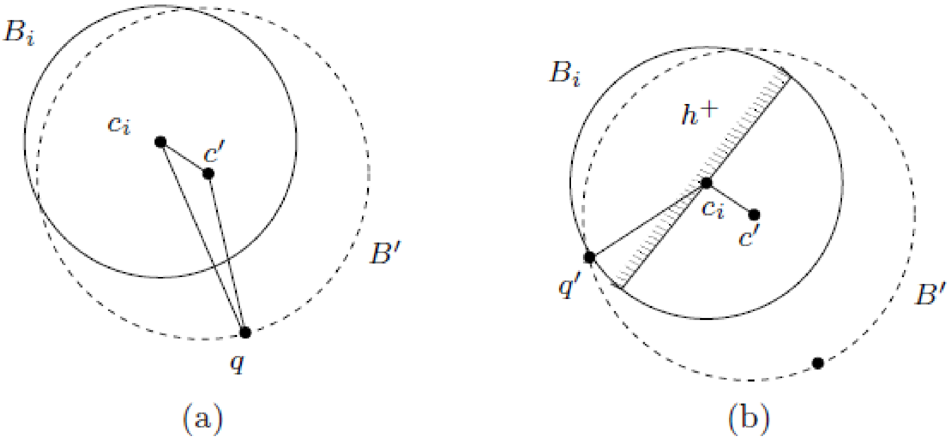


Figure.1[AS]

We see two different cases

a. $||c_i c'|| \leq 2\epsilon r_i/3$ (Figure.1(a)), then

$$\begin{aligned} r' &\geq ||c' q|| \\ &\geq ||c_i q|| - ||c' c_i|| \\ &\geq (1+\epsilon)r_i - 2\epsilon r_i/3 \\ &\geq (1+\epsilon^2/8)r_i. \end{aligned}$$ { if $\epsilon < 8/3$ then $\epsilon/3 > \epsilon^2/8$ }

b. $||c_i\, c'|| > 2\epsilon r_i/3$ (Figure.1(b)), then

then let h be the hyperplane passing through ci with the direction c_ic' as its normal, and let h^+ be the halfspace, bounded by h, that does not contain c′. By Lemma 1, there is a point $q' \in K_i \cap h^+$ that is at a distance r_i from c_i. Therefore

$r' \geq ||q'\, c'||$

$\geq (\, ||c_i\, c'||^2 \;+\; ||q'\, c_i||^2\,)^{1/2}$ distance

{projection of q′ on hyperplane make right angle triangle and q'>its projection}

$\geq ((2\epsilon r_i/3)^2 + (r_i)^2\,)^{1/2}$

$\geq (1 + \epsilon^2/8)r_i$. {as $16(16\text{-}9)/9 > \epsilon^2$ }

First property is proven by Lemma 2

If for every $p \in A$, there is an $i \leq u$ such that $p \in (1{+}\epsilon)B_i$, then the set K does not change, and all three properties continue to hold. Hence, assume that there is a $p \in A$ such that $p \notin B_i$ for all i≤u. By Lemma 2, property-1 continues to hold after each update. Note that if $p \notin (1 + \epsilon)B_i$ for all $1 \leq i \leq u$, then Update computes a (1+ϵ/3)-MEB(K∪A). Since K^* is the only subset added to K, $K \subseteq (1 + \epsilon)B^*$ and K^* is added at the end of the sequence, $Ki \subseteq (1{+}\epsilon)B^*$. Thus property-2 holds after each Update. Note that a prefix K_D is deleted from K, so property-3 may be violated. However, the next two lemmas prove that it is also satisfied.

Lemma 3: For i < j, $c(B_i) \in (1 + \epsilon)B_j$.

Proof is as follows:

Suppose, on the contrary, that $c(B_i) \notin (1{+}\epsilon)B_j$. Let h be a halfspace that contains $c(B_i)$ but $h \cap (1 {+}\epsilon)B_j = \emptyset$. By Lemma 1, h contains a point $p \in K_i$, but $p \notin (1{+}\epsilon)B_j$, which contradicts the fact $K_i \subset (1{+}\epsilon)B_j$ (by property-2). Hence, $c(Bi) \in (1 + \epsilon)B_j$

Lemma 4: For all $K_i \in K_D$, $(1 + \epsilon)B_i \subseteq (1 + \epsilon)B^*$.

Proof is as follows:

Let $c^* = c(B^*)$, $r^* = r(B^*)$, $c_i = c(B_i)$, and $r_i = r(B_i)$. Let $K_i \in K_D$, then $r_i \leq \epsilon r^*/4$. Since $Ki \subset (1 + \epsilon/3)B^*$, the proof of Lemma 3 implies $c(B_i) \in (1 + \epsilon/3)B^*$. For any point $x \in (1 + \epsilon)B_i$,

$||x\, c^*|| \leq ||c^*\, ci|| + ||c_i\, x||$

$\leq ||c^*\, c_i|| + (1 + \epsilon)r_i$

$\leq (1 + \epsilon/3)r^* + \epsilon(1 + \epsilon)r^*/4$

$\leq (1 + \epsilon)r^*$.

Therefore $q \in (1{+}\, \epsilon)B^*$ and thus $(1{+}\, \epsilon)B_i \subseteq (1{+}\, \epsilon)B^*$.

Lemma 4 immediately implies that for any $K_i \in K_D$, if there is a point $q \in (1{+}\, \epsilon)B_i$ then $q \in (1{+}\, \epsilon)B^*$. Hence, for every $q \in S$, there is an i such that $q \in (1 + \epsilon)B_i$, implying property-3.

2.4.1.1 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 $r_u \geq (1+\epsilon^2/8)^u r_1$, which gives $u \leq \lceil \log_{1+\epsilon 2/8}(4/\epsilon) \rceil = O((1/\epsilon^2) \log(1/\epsilon))$.

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

For any $0 < \epsilon \leq 1$, an ϵ -blurred ball cover of size $O((d/\epsilon^3) \log(1/\epsilon))$ of a stream of points can be maintained.

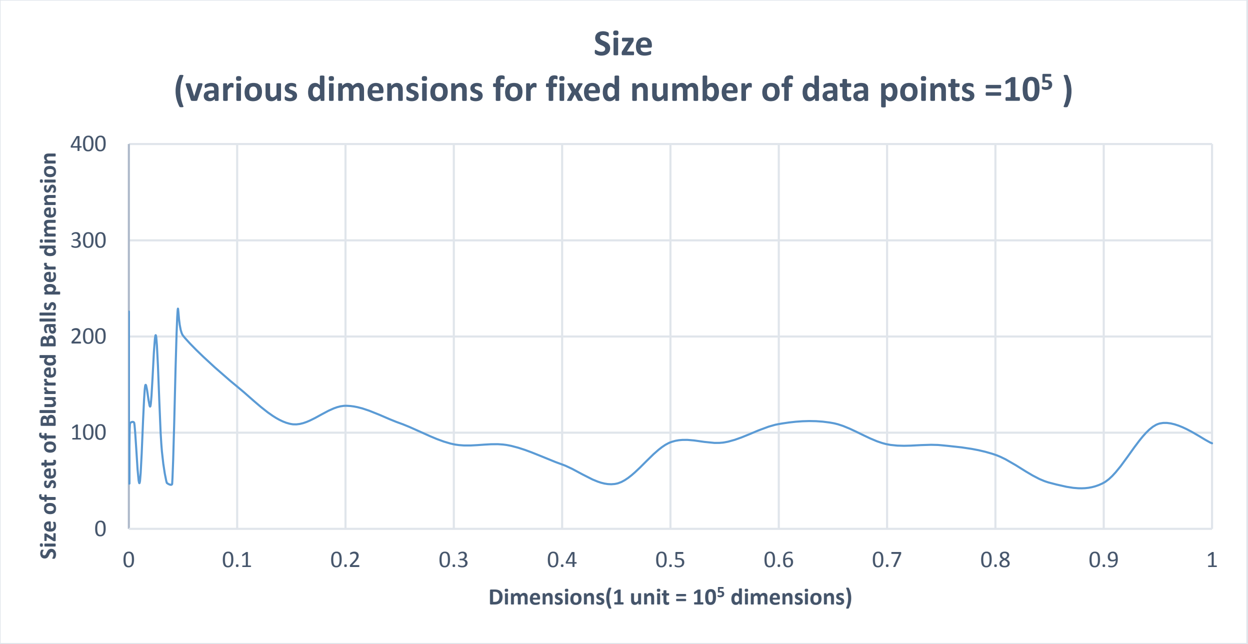
2.4.2 Implementation and Results of Update Procedure

Implementation of above algorithms is done in C and results are calculated for various values of ϵ. Results are compared both with respect to number of data points and dimensions.

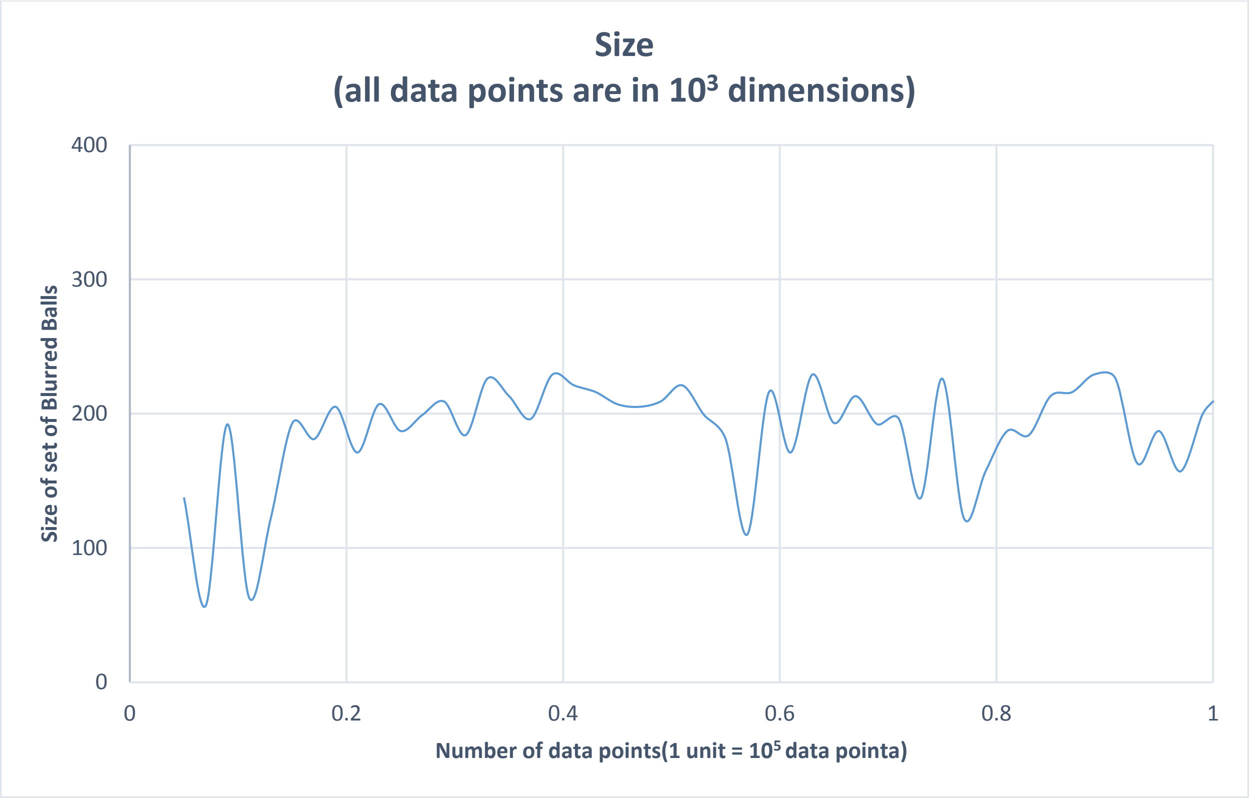
2.4.2.1 $\epsilon = 0.1$

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

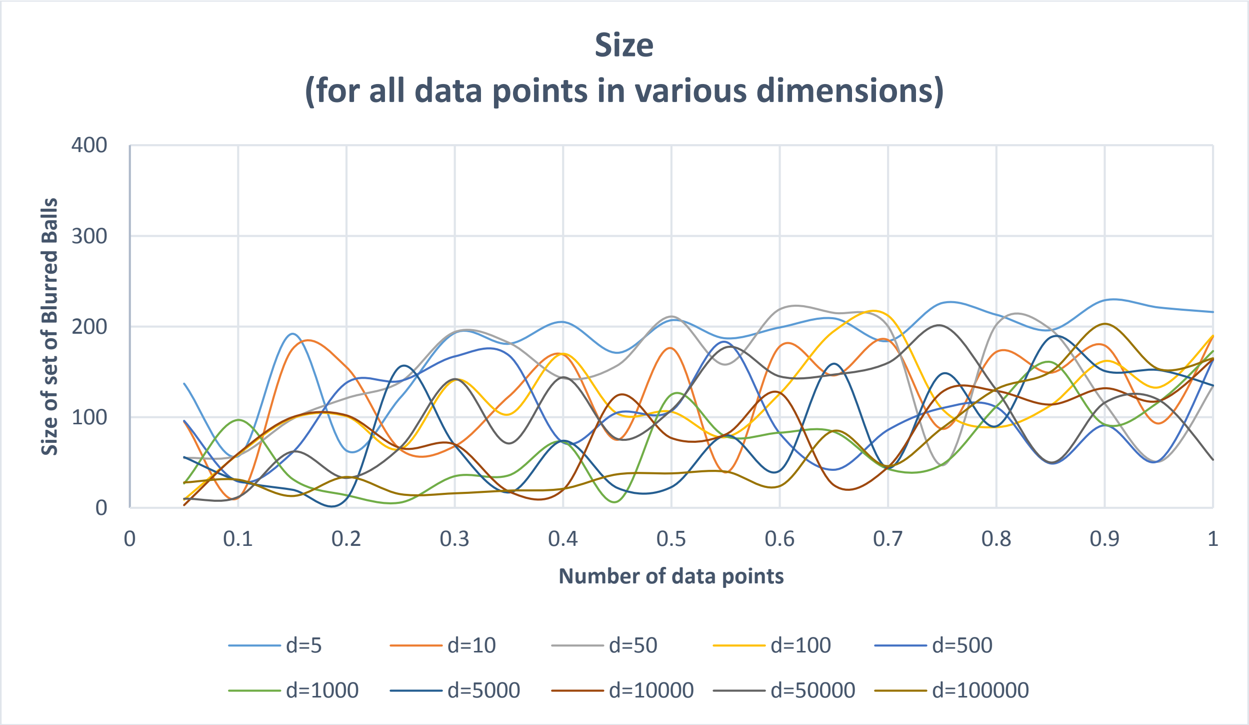
To compare, results have been normalized in dimensions



Plot.5


Size
(all data points are in 10^3 dimensions)
Size of set of Blurred Balls
400
300
200
100
0
0
0.2
0.4
0.6
0.8
1
Number of data points(1 unit = 10^5 data pointa)

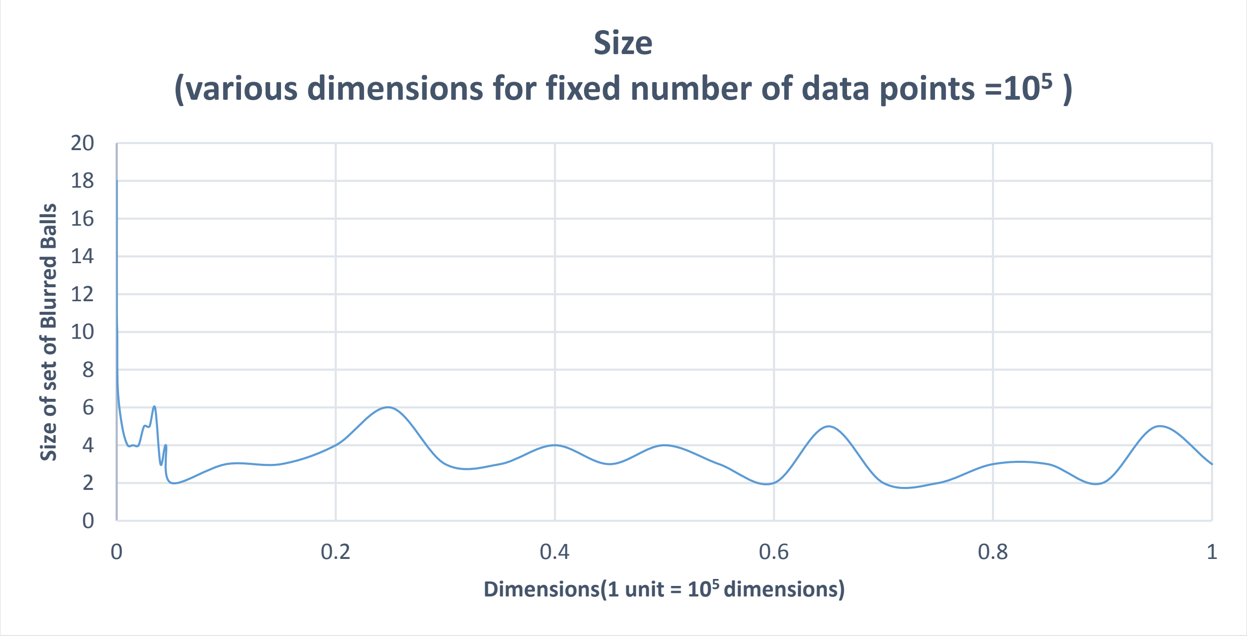
Plot.6


Size
(for all data points in various dimensions)
Size of set of Blurred Balls
400
300
200
100
0
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
Number of data points
d=5
d=10
d=50
d=100
d=500
d=1000
d=5000
d=10000
d=50000
d=100000

Plot.7

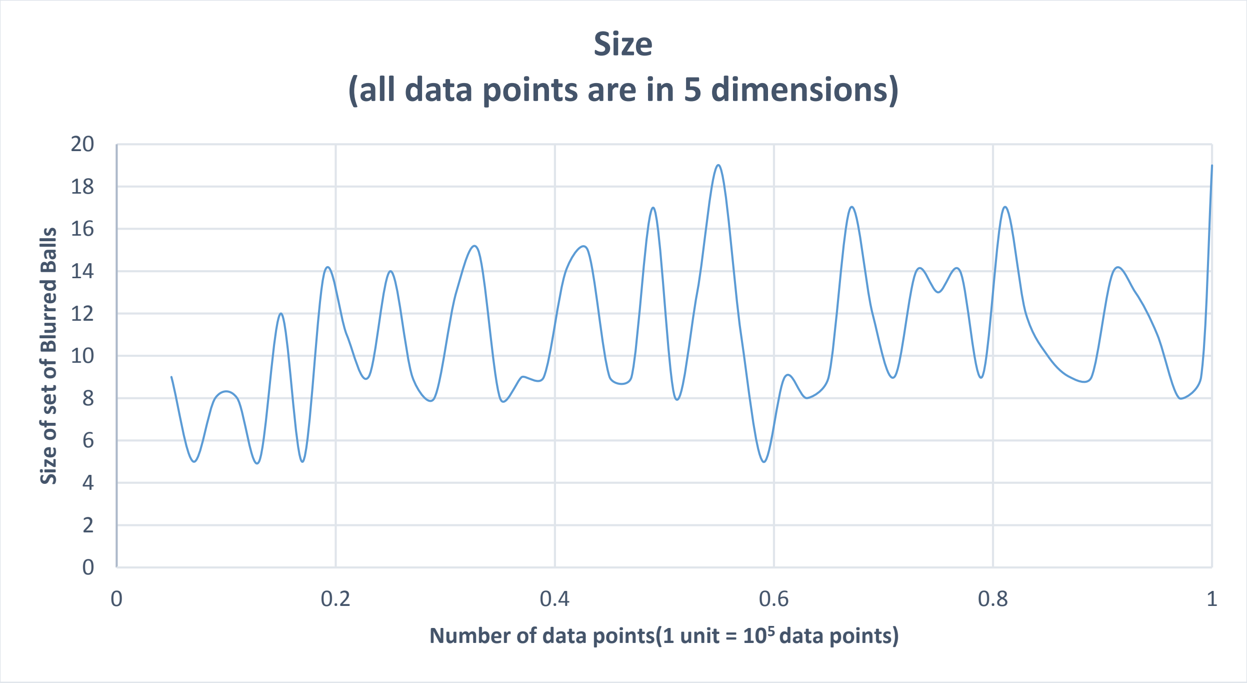
2.4.2.1 $\epsilon = 0.3$

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

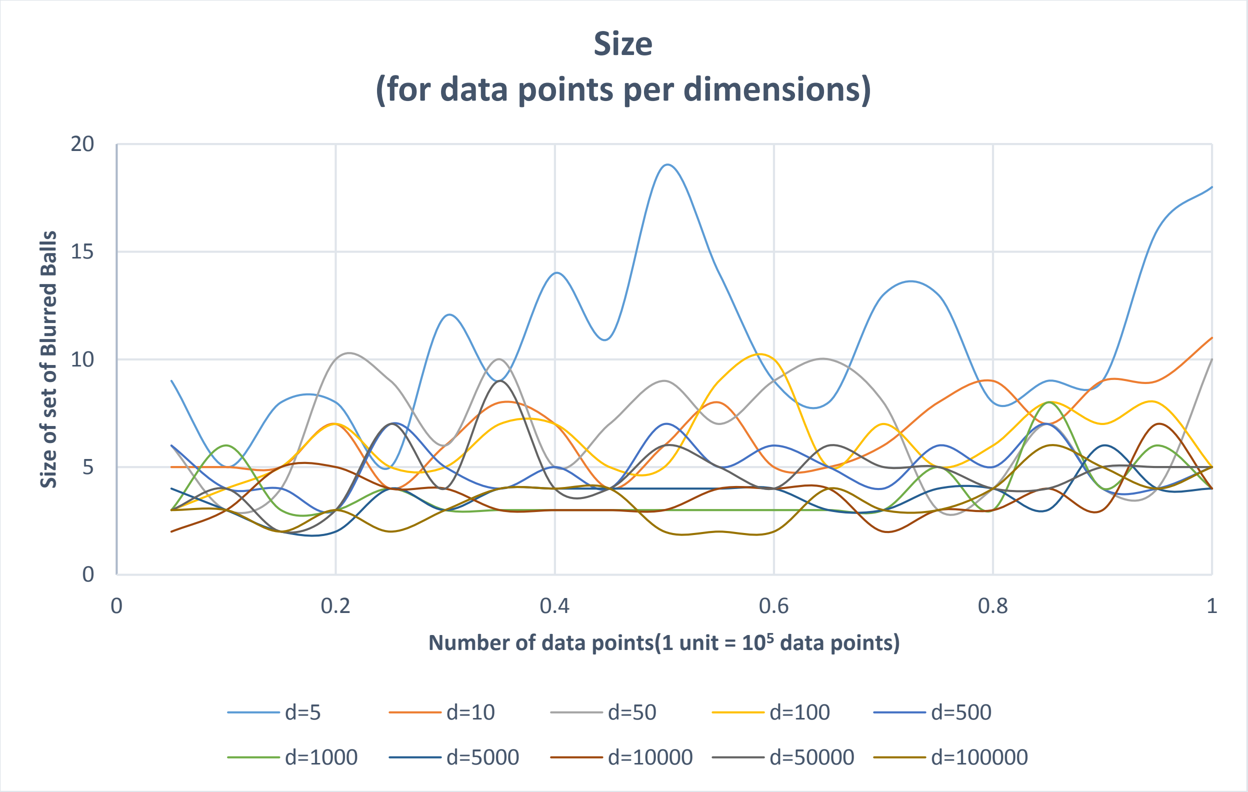


Plot.8

Plot.9



Plot.9


Size
(for data points per dimensions)
Size of set of Blurred Balls
20
15
10
5
0
0
0.2
0.4
0.6
0.8
1
Number of data points(1 unit = 10⁵ data points)
d=5
d=10
d=50
d=100
d=500
d=1000
d=5000
d=10000
d=50000
d=100000

Plot.10

2.4.5 Use Blurred-Ball to find MEB

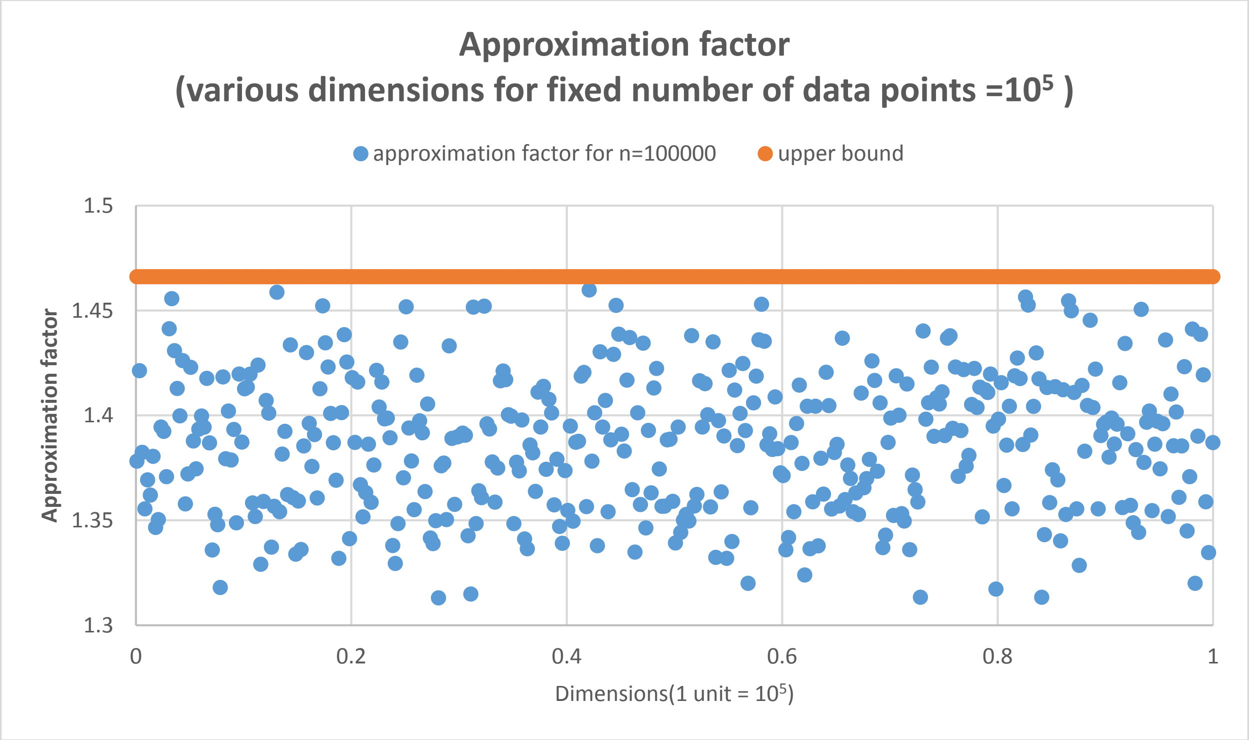
For a stream S of points, we maintain a (ϵ/9)-blurred ball cover K of S using above Update procedure. K is updated whenever a new point arrives. Let B = {B_1, . . . ,B_u}. Let B* = MEB(B). We return (1+ϵ/3)B* which can be computed in time O(1/$ϵ^5$) [KMY].

We will directly use the results proved in [KMY] that returned ball has maximum size ((1 + √3)/2 + ϵ/3)r, where r is r(MEB(S)).

2.4.6 Implementation and Results

2.4.2.1 ϵ = 0.1

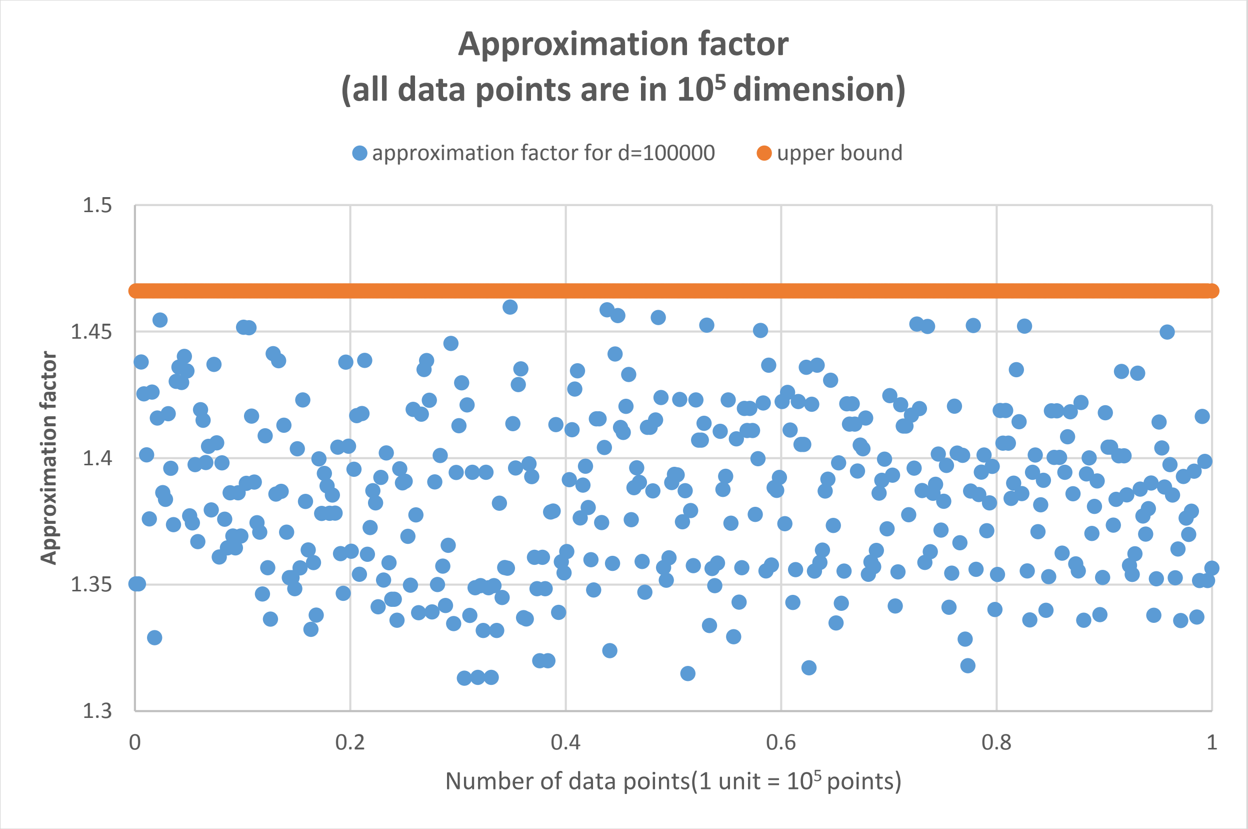
(1+√3)/2 + ϵ = 1.466



Plot.11

To check the effect of number of data points

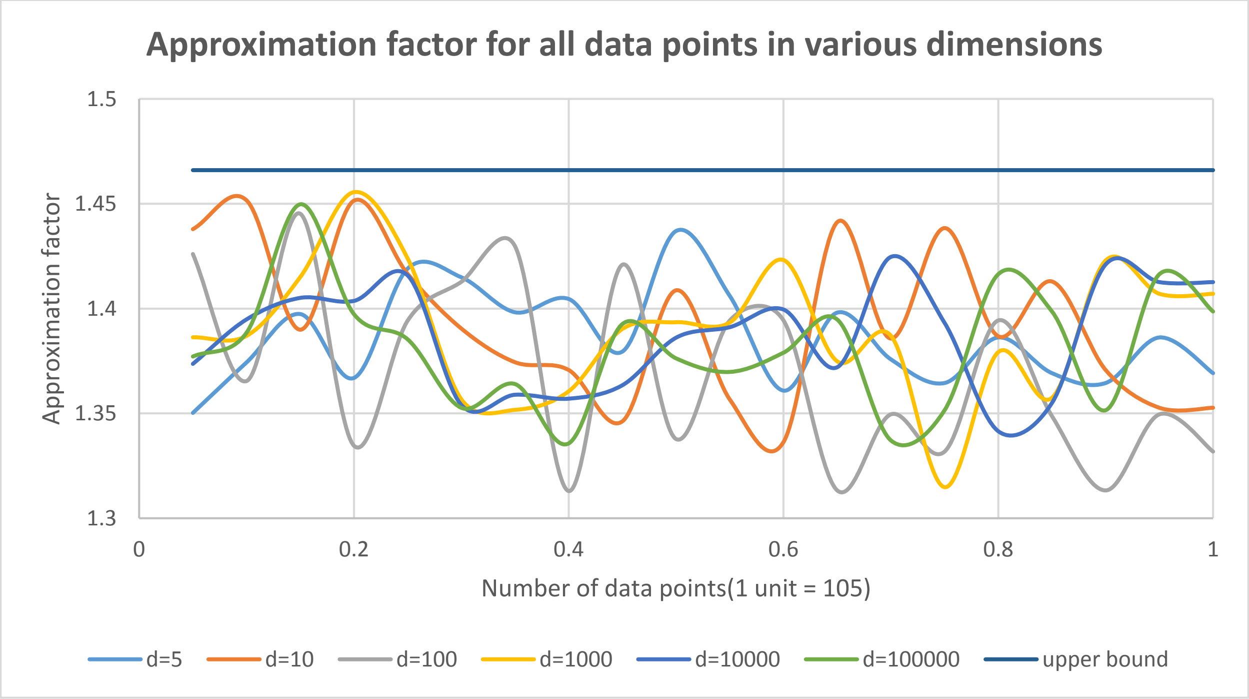
Number of data points varies from 10 to 10^5 in fixed 10^5 dimension



Plot.12

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)



Plot.13

We can see that

No instance crossed the upper bound.

None went below a certain value

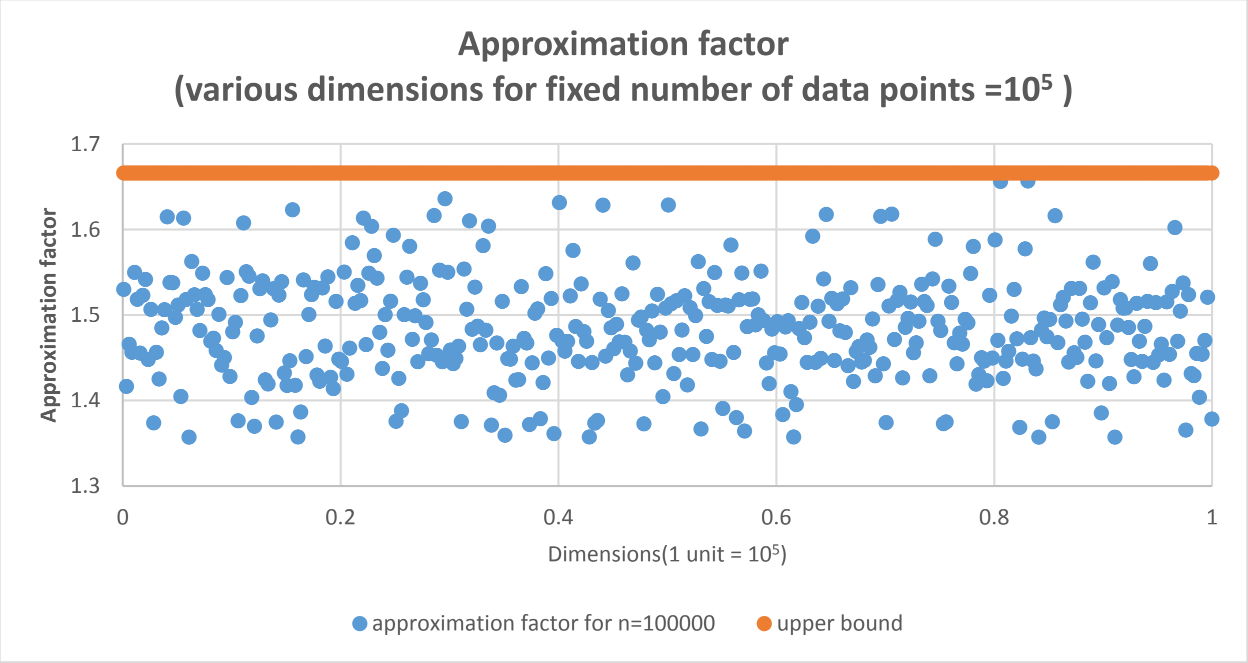
Similar results were found for different values of epsilon

2.4.2.1 ϵ = 0.3

(1+√3)/2 + ϵ = 1.666

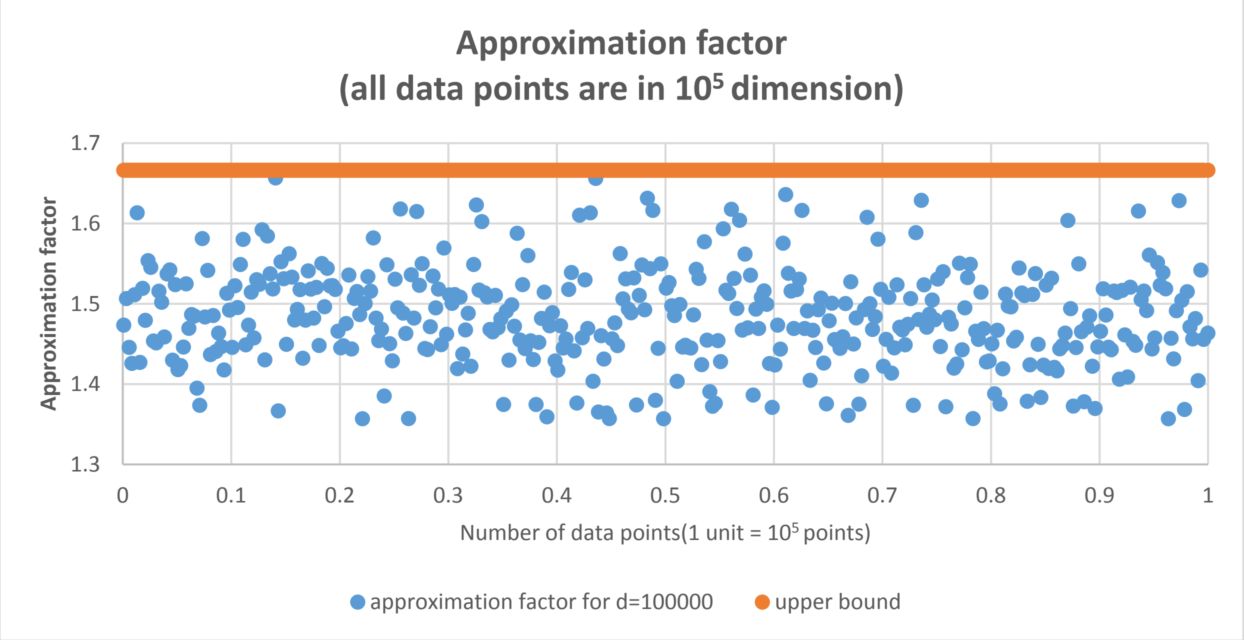
Plot to check the effect of dimensions on data points

Dimensions varies from 10 to 10^5 for fixed 10^5 data points

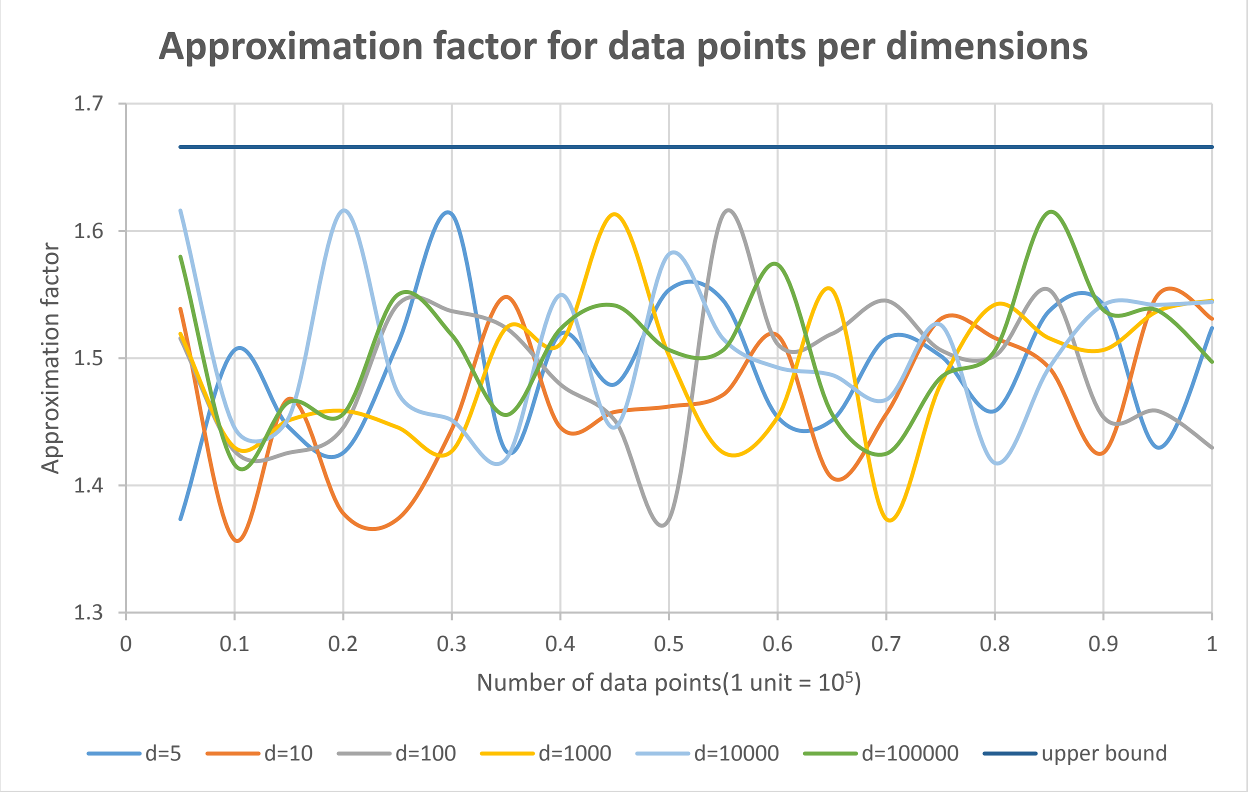


Plot.14

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



Plot.15



Plot.16

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

[14] T. M. C. Timothy M. Chan, "A Simple Streaming Algorithm for Minimum Enclosing Balls," *CCCG 2006, Kingston, Ontario,,* p. 4, 2006.