**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,

April,2014 Department of Computer Science & Engineering,

Guwahati. Indian Institute of Technology Guwahati, Assam.

**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].

( iv )

**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.

1.

**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. Number of vertices in graph
2. Number of edges in graph
3. , where is some constant.

**2.1 Semi-Stream**

In this model, the algorithm is given 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.

**2.1.2 Algorithm** [FKM+05b]

1:For each edgein the stream

2: **if** and are in different components, **do**

3: 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.

4: **else**, **do**

5: add to the MST for the component (creating a cycle) and remove the heaviest edge on the cycle created.

6: 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, so it takes space (which is equals to 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.

3.

Plot.1

Plot.2

4.

**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 105 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 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 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.

5.

**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 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

**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 listof the vertices followed by the listof the vertex-pairs representing the edges. Since the graphs we study are undirected, contains both pairsandfor an edge betweenand.

While bothandare 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, we produce a streamthat contains pairs of node namesand random distinct numbers, e.g. 3logn-bit random numbers:

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

1. First, we sort the pair of streamsand, so that the pairsare directly followed by all edges whose first component is:

In one pass on this stream, we can produce a new stream of edges, where for the first component of each edge, is replaced by the corresponding:

1. Now we sortand, so that the pairsappear right before all edges whose second component is equal to :

1. In this stream, the numberassigned to a nodeoccurs right before the list of numbers assigned to the nodes incident to. Thus, in one linear pass, we can determine the smallest number among them for each, which yields a streamof nodeswith their new labels:

3: Now check if then declare it as **Final Label** else

If not equal then and repeat this until we find stream and 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 ([ADR+04]). Use of random numbers make expected number of iteration as .

Every intermediate stream generated is of, 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 103. We have tested the algorithm for number of edges starting from linear,, to its maximum possible value that is 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 iterations and dense graph (nearly complete) takes iterations. But on an average a graph which is not the above two takes 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.

1:Sort the edges by increasing weights.

2: the lighter half edges.

3: Compute the connected components of using above algorithm.

4: *.*

5: **for** each connected components of, **do**

6: Compute MST recursively (call itand add their weights.

(

7: Now treat each components obtained in previous steps as a *node* of graph and find the edges connecting these *nodes*.

8: Compute the MST of this new *graph(call it* and add its weight.

9: return.

**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 104 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]

1:Ball

2:*.c* *pₒ*

3:*.r*

4: **For** each point *p* in the input stream *P* **do**

5: **If** *p* is outside *B* **do**

6:*r* i-1 *r*

7:*c* i-1 *c*

8:i *p c*i-1 *r*i-1

9:*c**p c*i-1

10:*r**r* i-1  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 ∈ Rd 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+ϵ)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 <ϵ ≤ 1, an ϵ -blurred ball cover of a set *S* of *n* points in *Rd*, denoted by *K* = *K(S)*, is a sequence <*K1, K2, . . . , Ku*>, where each *Ki* ⊆ *S* is a subset of O(1/ϵ) points that satisfies the following three properties;

Let *Bi* = MEB*(Ki)* and *K* =

1. *r(Bi+1)* ≥ (1 + ϵ2/8)*r(Bi)* 1 ≤ i < u
2. *Ki* ⊂ (1+ϵ)*Bj* i < j
3. *p* ∈ *S*, ∃i ≤ u such that *p* ∈ (1 + ϵ)*Bi*.

**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 ⊂ *Rd* and value ϵ > 0, a *core-set S* is a set of points S ⊂ Psuch 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 + ϵ, 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/ϵ). 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⊂Rd, computes *K(S ∪ 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 < ϵ ≤ 1 and returns a set *G* ⊆ *Z* of O(1/ϵ) points and *B = MEB(G)* such that Z ⊂ (1 + ϵ)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 *Bi ‘*s then the Update procedure invokes Approx-MEB on *K ∪ A* with approximation ratio ϵ/3. Let *K∗ ⊆ K ∪ 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 *Ki*’s for which *r(Bi) ≤ ϵr(B∗)/*4.

**Update(K,A)** [AS]

1:if ∃p ∈ A, ∀i ≤ u, p does not belong to (1 +ϵ)Bi then

2: *K∗,B∗ := Approx-MEB((K ∪ A), ϵ/3)*.

3: KD := {Ki | r(Bi) ≤ ϵr(B∗)/4}

4: *K := (K \ KD) օ <K∗>*

5: 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 B. [AS]

This is the property that every MEB satisfies.

**Lemma 2:** For all i < u, r(Bi+1) ≥ (1+ϵ2/8)r(Bi).

Proof is as follows:

Let B’ = MEB(K ∪ {q}).

Clearly r(Bi+1) ≥ r(B’) { as B’ is MEB which has }

Let r’ = r(B’), c’ = c(B’), ci = c(Bi), and ri = r(Bi).

We will prove that r’ > (1+ϵ2/8)ri. Which will prove our Lemma.



Figure.1[AS]

We see two different cases

1. ||ci c’|| ≤ 2ϵri/3 (Figure.1(a)), then

r′ ≥ ||c′ q||

≥ ||ci q|| − ||c’ ci||

≥ (1 + ϵ)ri − 2ϵri/3

≥ (1 + ϵ2/8)ri. { if ϵ < 8/3 then ϵ /3 > ϵ2/8 }

1. ||ci c’|| > 2ϵri/3 (Figure.1(b)), then

then let h be the hyperplane passing through ci with the direction cic’ 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′ ∈ Ki ∩ h+ that is at a distance ri from ci. Therefore

r′ ≥ ||q’ c’||

≥ ( ||ci c’||2 + ||q’ ci||2 )1/2 distance

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

≥ ((2ϵri/3)2 + (ri)2 )1/2

≥ (1 + ϵ2/8)ri. {as 16(16-9)/9 > ϵ2 }

First property is proven by Lemma 2

If for every p ∈ A, there is an i ≤ u such that p ∈ (1+ϵ)Bi, then the set K does not change, and all three properties continue to hold. Hence, assume that there is a p ∈ A such that p Bi for all i≤u. By Lemma 2, property-1 continues to hold after each update. Note that if p (1 + ϵ)Bi for all 1≤i≤u, then Update computes a (1+ϵ/3)-MEB(K∪A). Since K∗ is the only subset added to K, K ⊆ (1 + ϵ)B∗ and K∗ is added at the end of the sequence, Ki ⊆ (1+ϵ)B∗. Thus property-2 holds after each Update. Note that a prefix KD 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(Bi) ∈ (1 + ϵ)Bj .

Proof is as follows:

Suppose, on the contrary, that c(Bi) (1+ϵ)Bj . Let h be a halfspace that contains c(Bi) but h ∩ (1 +ϵ)Bj = ∅. By Lemma 1, h contains a point p ∈ Ki, but p (1+ϵ)Bj , which contradicts the fact Ki ⊂ (1+ϵ)Bj (by property-2). Hence, c(Bi) ∈ (1 + ϵ)Bj

**Lemma 4**: For all Ki ∈ KD, (1 + ϵ)Bi ⊆ (1 + ϵ)B∗.

Proof is as follows:

Let c∗ = c(B∗), r∗ = r(B∗), ci = c(Bi), and ri = r(Bi). Let Ki ∈ KD, then ri ≤ ϵr∗/4. Since Ki ⊂ (1 + ϵ/3)B∗, the proof of Lemma 3 implies c(Bi) ∈ (1 + ϵ/3)B∗. For any point x ∈ (1 + ϵ)Bi,

||x c∗|| ≤ ||c∗ ci|| + ||ci x||

≤ ||c∗ ci|| + (1 + ϵ)ri

≤ (1 + ϵ/3)r∗ + ϵ(1 + ϵ)r∗/4

≤ (1 + ϵ)r∗.

Therefore q ∈ (1+ ϵ)B∗ and thus (1+ ϵ)Bi ⊆ (1+ ϵ)B∗.

Lemma 4 immediately implies that for any Ki ∈ KD, if there is a point q ∈ (1+ ϵ)Bi then q ∈ (1+ ϵ)B∗. Hence, for every q ∈ S, there is an i such that q ∈ (1 + ϵ)Bi, implying property-3.

**2.4.1.1 Space Complexity**

Let ri = r(Bi). Update ensures that ru/r1 ≤ 4/ϵ. By (P1), ri+1 ≥ (1+ϵ2/8)ri.

Therefore ru ≥ (1+ϵ2/8)ur1 , which gives u ≤ ⌈log1+ϵ2/8(4/ϵ)⌉ = O((1/ϵ2) log(1/ϵ)).

Hence |K| ≤ |Ki| = O((1/ϵ3) log(1/ϵ)).

For any 0 < ϵ ≤ 1, an ϵ -blurred ball cover of size O((d/ϵ3) log(1/ϵ)) 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 ϵ = 0.1**

((1/ϵ)2log2(1/ϵ) = 332.192

To compare, results have been normalized in dimensions

Plot.5

Plot.6

Plot.7

**2.4.2.1 ϵ = 0.3**

((1/ϵ)2log2(1/ϵ) = 19.2996

Plot.8

Plot.9

Plot.9

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 = {B1, . . . ,Bu}. Let B∗ = MEB(B). We return (1+ϵ/3)B∗ which can be computed in time O(1/ϵ5) [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 105 in fixed 105 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 105 for fixed 105 data points

Plot.14

To check the effect of number of data points no. of data points varies from 10 to 105 in fixed 105 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. B˘adoiu 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. |