Clustering Geometric Data Streams

Jiří Skála[*]
Ivana Kolingerová[†]
University of West Bohemia

Abstract

Using recent knowledge in data stream clustering we present a modified approach to the facility location problem in the context of geometric data streams. We give insight to the existing algorithm from a less mathematical point of view, focusing on understanding and practical use, namely by computer graphics experts. We propose a modification of the original data stream k-median clustering to solve facility location which is the case when we a priori do not know the number of clusters in the input data. Like the original, the modified version is capable of processing millions of points while using rather small amount of memory. Based on our experiments with clustering geometric data we present suggestions on how to set processing parameters. We also describe how the algorithm handles various distributions of input data within the stream. These findings may be applied back to the original algorithm.

CR Categories: I.5.3 [Computing Methodologies]: Pattern Recognition—Clustering; I.3.5 [Computing Methodologies]: Computer Graphics—Computational Geometry and Object Modeling

Keywords: data stream, clustering, facility location, geometric data

1 Introduction

Data stream algorithms have been extensively studied in connection with databases and network statistics. However, there is not much research dedicated to geometric data streams. Since geometric models are growing larger and larger like those from Stanford's 3D Scanning Repository [Stanford 2007], data stream approach is becoming essential to process such models. By clustering we can significantly reduce the amount of data. Clusters can be then used to create a multiresolution model. Data stream clustering is done in a hierarchical way which gives a possibility to control the use of memory. The algorithm runs with just several megabytes of memory while processing millions of points.

In our paper we propose a method for clustering geometric data in a streaming fashion. Assuming we do not know how many clusters there are, we think of the clustering as a facility location problem. We present a modified k-median algorithm to solve the facility location. Data stream processing is performed in blocks. Semi-results from blocks are processed at higher levels. This leads to a hierarchy which offers a possibility to build a multiresolution model. Unlike many algorithms for large geometric data, our method can process unordered points without any pre-processing. Based on our experiments we give suggestions of algorithm parameter settings and describe how the method works with various input data.

Section 2 briefly introduces problem background. Section 3 presents current state of the art in clustering and data stream algorithms. Sections 4 and 5 describe the method in detail. Our modifications and improvements to the current algorithm are described in Section 6. Section 7 presents the experimental evaluation. In Section 8 we give a conclusion and suggest future work.

2 Background

2.1 Clustering

Clustering stands for a wide range of problems. It concerns grouping similar elements together to form clusters. Euclidean distance is most often used as a measure of similarity. Perhaps the most common is the k-centres or k-means clustering. In this case we divide the elements (also referred to as points) into exactly k clusters so that the sum of distances from each point to the corresponding cluster centre is minimized. Another formulation of the problem is the k-median or k-medoid clustering where we search for cluster centres only among input points.

However, not always we know the number of clusters k. This problem is known as the facility location. A facility can be understood as a cluster centre. Given a set of points we choose some of them and open a facility there. All other points are connected to the closest facility. The problem is then to determine at which points a facility should be opened.

Since we want to minimize the distance between each point and its facility, it would be best to open a facility everywhere. But that is not what we want. That is why we introduce a facility cost - a penalty for opening every facility. Since we assume no differences between facilities, we set the same cost for all of them. Then we have the sum of distances between points and facilities on one hand, and the sum of facility costs on the other hand. Facility cost determines the balance between the cluster size and the number of clusters. This leads to an overall clustering cost C which is the sum of expenses for opening facilities plus the sum of distances from points to their facilities.

$$C = k \cdot fc + \sum_{i=0}^{N-1} \| c_i - f_{ci} \| \tag{1}$$

where k is the number of open facilities, fc is the facility cost (the same for all facilities), N is the number of points and $\| c_i - f_{ci} \|$ denotes the distance of point c_i to its facility. Those points where a facility is opened are supposed to be assigned to themselves thus having zero distance from their facility.

2.2 Data Stream

Data stream is a sequenced set of data that can only be viewed in order; there is no random access possible. Moreover, the set is supposed to be too large to fit into main memory. The data may

[*]e-mail: jskala@kiv.zcu.cz
[†]e-mail: kolinger@kiv.zcu.cz ; Work has been supported by the Ministry of Education, Youth and Sports of the Czech Republic under the research program LC-06008 (Centre for Computer Graphics).

come online in some time intervals or even the whole set may be stored but in some slow external memory. So the data stream must be processed in pieces and during one or very few linear scans.

3 State of the art

Clustering has been studied in many areas from many points of view. An overview of clustering techniques may be found in [Jain et al. 1999].

Several approaches exist for the facility location; a nice overview can be found in [Shmoys 2000]. A fundamental approach is based on linear programming relaxation [Chudak and Shmoys 2004; Charikar and Guha 1999]. The facility location problem is formulated as a linear programming (LP). The LP relaxation is solved in polynomial time and gives an approximate solution to the original problem. There is a related method also based on LP [Charikar and Guha 1999; Chudak and Shmoys 2004]. It uses a primal-dual scheme. A dual linear program is solved which gives a solution to the primal problem.

Another approach uses a technique of local search [Charikar and Guha 1999; Korupolu et al. 1998]. A coarse initial solution is iteratively improved by the local search. In each step a point is chosen at random and it is determined whether it is profitable to open a facility there. If so, nearby points are reassigned to this new facility. If there are facilities with a low number of points, these facilities are closed and points reassigned elsewhere.

There are methods for clustering large databases. CLARANS [Ng and Han 1994] views the problem of finding cluster centres as a graph searching problem. Each set of centres is interpreted as a graph node. By replacing one of the centres we get to a neighbouring node. The task is to get to a node with the minimal clustering cost. This is done by traversing the graph. To limit the computational complexity, CLARANS does a randomized search, i.e. it inspects just a random sample of neighbours for each node. BIRCH [Zhang et al. 1996] uses another approach. For each cluster it keeps a Clustering Feature - a vector of summary information about the cluster. Clustering Features are organized in a CF tree. Leaf nodes represent particular clusters. Higher nodes represent a cluster formed by all the children subclusters. CURE [Guha et al. 1998] uses a hierarchical clustering where small clusters are merged into larger ones. CURE keeps a well scattered sample of points for each cluster. It is a compromise between all-point and centroid-based cluster representation. Samples are then used to identify nearby clusters that should be merged together. Nevertheless, forenamed methods do not perform true data stream processing. They require the whole data set to be in the main memory.

[Muthukrishnan 2003] gives a nice overview of data stream algorithms. Data stream clustering is extensively studied by Guha et al. [Guha et al. 2000; O'Callaghan et al. 2002; Guha et al. 2003]. Using a hierarchical approach based on local search they solve the k-median problem. Data stream processing is performed in blocks. Semi-results from each block are maintained as a higher level stream which is continuously reprocessed in the same way, forming higher levels. [Meyerson 2001] deals with online facility location. For each newly arrived point the algorithm measures the distance to the nearest existing facility. It is then decided whether to connect the point to that facility or whether to open a new facility at the point. Charikar et al. propose algorithms that can handle outlier points in input data [Charikar et al. 2001]. The main idea is that some small fraction of points (outliers) may be left unassigned to any cluster. A penalty is assigned to every outlier point and the sum of penalties is added to the overall clustering cost (for the facility location it is Equation 1).

Stream processing of geometric data is extensively studied by Isenburg et al. Their research ranges from streaming formats for polygon meshes [Isenburg and Lindstrom 2005], streaming compression of geometric models [Isenburg et al. 2006a], to streaming computation of Delaunay triangulation [Isenburg et al. 2006b]. Other related works include polygonal models simplification [Lindstrom 2000] or approximate Voronoi diagrams [Sharifzadeh and Shahabi 2004]. Stream processing of points is addressed in [Pajarola 2005]. This work introduces so called stream operators which are applied while sweeping sorted data. Perhaps the first use of clustering for multiresolution models can be found in [Rossignac and Borrel 1993]. It uses a simple clustering to create approximations of 3D polyhedra for rendering complex scenes. More recent research in clustering geometric data streams is described in [Frahling and Sohler 2005] which is concerned with dynamic geometric data streams. In such a case we have a set of points and the stream consists of insert/delete operations among these points.

4 The clustering algorithm

As stated in Section 1 we will be concerned with the facility location problem. Our solution employs the Local Search algorithm proposed by Charikar and Guha [Charikar and Guha 1999]. First an approximate initial solution is generated. It is then iteratively refined by a series of local search improvements.

First suppose we already have some initial solution. Local search improvements can start. We take a point at random and we ask a question: What if we open a facility here? Would it be beneficial? First we will have to pay for opening the facility (if it is not already open). We then inspect all other points and compare the distance to their current facility with the distance to the new facility candidate. If the candidate lies closer, the point is reassigned to it, sparing some connection expenses. To limit computational complexity, any point can be reassigned only to the facility candidate. The time complexity of this first phase of one local search step is $\mathcal{O}(N)$, where N is the number of all points (we compute the distance of the facility candidate to all other points).

After that some facilities may contain just a few points. If we reassigned all of these remaining points (even if the new facility is farther), we would be able to close their old facility and spare the facility cost for it. But we must be careful whether the facility cost spared will overweight the expenses for reassigning points to a farther facility. This second phase has $\mathcal{O}(N)$ time complexity too. It could be computed simultaneously with the first phase.

So for some point p taken at random we determine whether it would be profitable to open a facility there. This is expressed by the *gain* function. Let us first define a *distance spare* ds_i as the distance we spare by reassigning point c_i to p. It is the difference between distances to the current facility and to the facility candidate p. If the difference is negative (current facility lies closer then p) we set $ds_i = 0$. Next we define a *close spare* cs_j as a cost we can spare by closing facility f_j. It is the facility cost minus expenses for reassigning all points from f_i to p. Again if cs_j is negative (we cannot spare anything) we set $cs_j = 0$. The gain function is then computed according to Formula 2

$$gain(p) = -fc + \sum_{i=0}^{N-1} ds_i + \sum_{j=0}^{M-1} cs_j \qquad (2)$$

where fc is the facility cost (zero if there is a facility already open at p), N is the number of all points and M is the current number of facilities. As stated before, computing function gain takes $\mathcal{O}(N)$ time.

Now to the initial solution. A very coarse one is sufficient, since local search will improve it quickly. All facilities have the same cost, so we use a simple algorithm proposed in [Meyerson 2001]. Points are taken in random order. At the first one a facility is always created. At every other point a facility is opened with probability d/fc, where d is the distance of the current point to the closest facility and fc is the facility cost. If $d/fc > 1$ we set the probability to 1.

The described local search technique is repeated $N \log N$ times. The number of iterations is derived in [Charikar and Guha 1999]. The initial solution can be generated in $\mathcal{O}(N^2)$ time. Function gain has $\mathcal{O}(N)$ time complexity and is evaluated $N \log N$ times. So the complete clustering algorithm has an overall time complexity of $\mathcal{O}(N^2 \log N)$.

5 Clustering a data stream

For clustering a data stream we use a hierarchical approach proposed in [Guha et al. 2000]. The algorithm inputs a block of points from the data stream and performs a clustering on it. Resulting facilities are given a weight according to the number of points assigned to them. Thus clusters containing more points have more importance. Weighted cluster centres are then passed to a higher level. Remaining points are discarded and the algorithm proceeds with another block from the data stream.

Facilities at higher levels are treated as weighted points and are also processed in blocks. When enough points gather to fill up an entire block, they are clustered again. This time the distance of a point to its facility is multiplied by the point weight. Resulting cluster centres are given a weight equal to the sum of weights of points assigned to them. Weighted facilities are again passed to a higher level. Figure 1 illustrates the hierarchical processing. Black dots indicate cluster centres in particular blocks. Blocks in the data stream are delimited by bold lines.

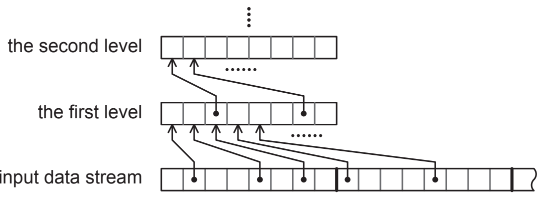

Figure 1: Hierarchical clustering.

The number of levels l required to process the entire data stream can be computed as

$$l = \frac{\log(N/m)}{\log(m/k)} \tag{3}$$

where N is the number of all points, m is the block size and k is the average number of clusters in one block. The equation was presented in [Guha et al. 2000] without any further explanation. Let us show how it can be derived. At the zero level the data stream is divided into N/m blocks. These blocks will be clustered, resulting in $N/m \cdot k$ facilities divided into $N/m \cdot k/m$ first-level blocks. The situation repeats at higher levels until resulting l-level facilities fit into a single block. We can write this as

$$N/m \cdot k/m \cdot k/m \cdot \ldots \cdot k/m = 1 \tag{4}$$

where k/m repeats l-times. After rearrangement

$$N/m = m/k \cdot m/k \cdot \ldots \cdot m/k = (m/k)^l \tag{5}$$

Taking a logarithm of Equation 5 we get

$$l = \log_{m/k}(N/m) = \frac{\log(N/m)}{\log(m/k)} \tag{6}$$

6 Our modifications and improvements

6.1 Making Local Search More Local

We may speed up the evaluation of function gain by limiting the number of points that need to be inspected. Given a facility cost fc, any point can be connected to a facility at most fc far away. Otherwise it is cheaper to open a new facility at that point. Let us define the *influence area* of facility f to be the circle with centre f and radius fc. All points connected to f must then lie within its influence area.

When computing the gain function we can inspect just those points whose distance from the facility candidate is at most fc. How do we find them? Any cluster can contain just points lying within radius fc from the cluster centre. So when we take all facilities in a $2 \cdot fc$ radius around the facility candidate, and examine all points connected to those facilities, we can be sure that we inspected all necessary points.

The situation is illustrated in Figure 2. Facilities are shown as dots with an influence area drawn as dotted circle. The facility candidate is designated black. We need to inspect all points that may lie within its influence area IA. Such points can only belong to facilities whose influence area overlaps with IA. Such facilities (shown as diamonds) lie within the dashed circle with radius $2 \cdot fc$.

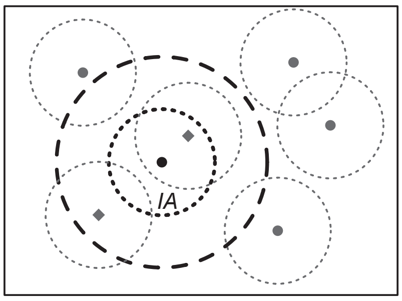

Figure 2: Finding points that may lie within the facility candidate's influence area.

For weighted points the situation is a bit more complicated because distances are multiplied by point weights. So a point with a small weight can be connected to a distant facility, while a point with a high weight should be connected to a nearby facility. But the above idea can still be used. For each facility we find the point with minimal weight w_{min}. The influence area radius is then fc/w_{min}. So we need to inspect all points assigned to facilities lying at most $2 \cdot fc/w_{min}$ away from the facility candidate. Note that w_{min} could be different for each facility.

6.2 From k-median to Facility Location

Cited papers deal with the k-median problem. They compute facility location repeatedly and using binary search they find such facility cost that yields exactly k clusters. Our modification computes the clustering just once. Solving the facility location itself seems to be just part of the original method, but it is not so simple. Since we have no k we must choose a suitable facility cost to get natural clusters. We suggest setting the facility cost for each block of data equal to the diagonal of bounding box. This is discussed in detail in Section 7.1.

Another problem brings clustering at higher levels. Points have weights so all distances are multiplied by some (possibly large) numbers. If we perform the clustering as usual, a facility would be opened at almost every point because weights make them several times farther from each other. We could increase the facility cost but point weights will grow higher with increasing level and we may encounter numerical problems. Instead of scaling the facility cost we decided to introduce weight normalization.

Points at level zero have unit weight. We would like to keep weights around one also at higher levels. Therefore we divide all weights by their average. The average of new weights will be one, exactly as we wanted. It is important to do the normalization of all the points in a block at the same time. That means not earlier then the block is full. Normalizing weights right after clustering a lower level block (before passing points to higher level) is wrong. Each block may have a different number of clusters so the average weight may also vary. Thus points from different blocks would not be normalized equally.

7 Experiments and results

It this section we present experiments made with clustering geometric data. We discuss how the result depends on parameter settings and on the distribution of points in the stream. We give recommendations on how to set parameters to get a good clustering.

The algorithm was implemented in C# 2.0. Experiments were done on Intel Pentium 4 3.2 GHz with 2 GB RAM and SATA HDD, running Windows XP Professional. All measured times include I/O operations. Memory requirements were measured using the Performance Monitor in Windows XP.

7.1 Setting the Facility Cost

The facility cost determines how strongly the data will be clustered. High setting means an aggressive clustering resulting in a lower number of large clusters. Low setting will cluster just moderately producing many smaller clusters.

Some experiments are usually needed to find a facility cost that best fits your needs. It is a counterbalance to point distances. We need a small facility cost for clustering points in a unit square, and a high one for points in $[0; 10^6]$ interval. To avoid time consuming normalization, the facility cost must be derived from the range of point coordinates. Coordinate magnitude does not matter because we only care about point distances. We suggest setting the facility cost for each block equal to the diagonal of bounding box. It mostly produces good results. You can double the facility cost for a stronger clustering or divide it by two (or even by four) to get a moderate clustering.

Of course each block may have a different bounding box. So the clustering will be performed in each block with a different facility cost. But this is not a problem since we use weights. For a low facility cost we get many facilities with a low weight. High facility cost produces few facilities with big weight. Figure 3 shows an example of clustering with a facility cost set to double and half the diagonal respectively. Numbers show facility weights (before normalization).

Remember that a strong clustering reduces the amount of data faster. The algorithm may then run with fewer clustering levels, having lower memory requirements and shorter execution time. Indeed, the opposite holds for a moderate clustering.

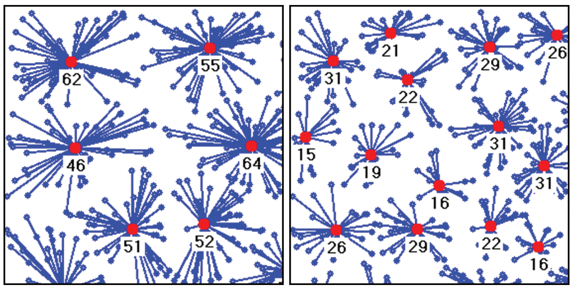

Figure 3: Clustering with a double facility cost (left) and a half facility cost (right). Figures cropped.

7.2 Input Point Distribution

Most authors concerned with large geometric data often rely on that input data will be more or less ordered. Our algorithm can handle unordered data as well.

If points arrive in order the algorithm processes them successively cluster by cluster. Transitions between clusters are generally handled correctly. The situation is illustrated in Figure 4. The dataset contains 2200 points in four groups (550 points each). It was processed with facility cost set equal to the bounding box diagonal, block size 750. Frames a)–c) show three blocks at level zero. Resulting facilities are passed to a block at the first level shown in Frame d).

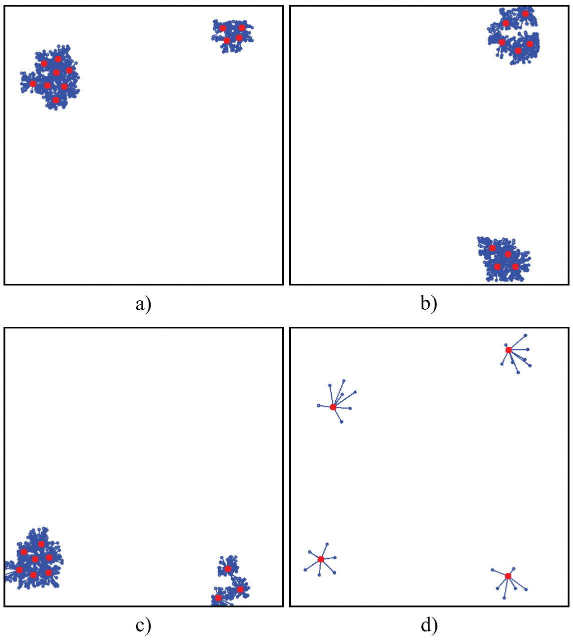

Figure 4: Clustering ordered data.

If points are scattered in the stream, and they come in a rather random order, it does not mean any trouble. The algorithm will process several points from different clusters at once. These points form something like cluster fragments which will be merged at higher levels. You can see an example in Figure 5. This is the same data as in Figure 4, they were just shuffled. Processing parameters were also the same. Frames a)–c) show three blocks at level zero. Resulting facilities are passed to a block at the first level which is shown in Frame d).

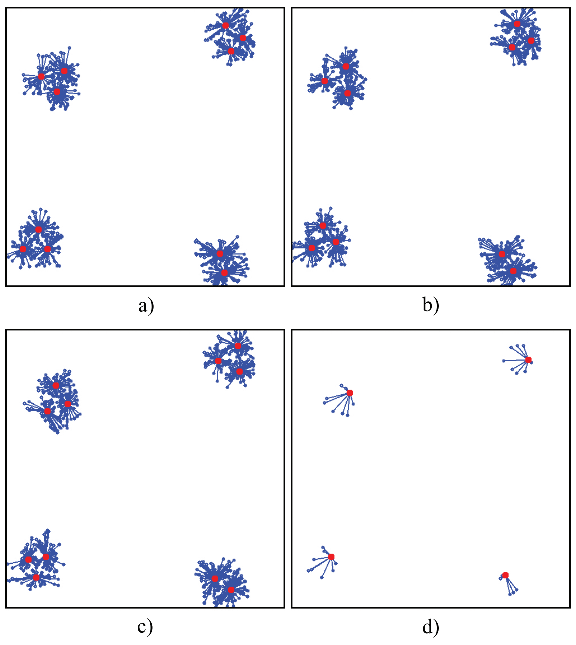

Figure 5: Clustering unordered data.

7.3 The Block Size

The clustering also depends on the size of block in which input data are processed. Block size basically affects execution time, the amount of memory used and also the clustering result.

Let N be the number of all points, m be the block size and k be the average number of clusters in each block. The number of all blocks at all levels will be

$$\begin{aligned} & N/m + N/m \cdot k/m + N/m \cdot k/m \cdot k/m + \ldots = \\ = \; & N/m \cdot [1 + k/m + (k/m)^2 + \ldots] = \\ = \; & N/m \cdot 1/(1-k/m) = N/m \cdot m/(m-k) = \\ = \; & N/(m-k) \end{aligned} \tag{7}$$

As proved in [Charikar and Guha 1999], $m \log m$ local search iterations are necessary for each block. So the total number of iterations over all blocks will be

$$N/(m-k) \cdot m \log m \tag{8}$$

Because k is proportional to m, we can write

$$N/(m - c_1 \cdot m) \cdot m \log m = c_2 \cdot N \cdot \log m \tag{9}$$

where c_1, c_2 are some constants. So by decreasing block size m, the number of iterations necessary to process the whole data set also decreases.

Lowering the block size also decreases memory requirements. Let l be the number of all levels. The amount of memory required is proportional to

$$m \cdot l = m \cdot \frac{\log(N/m)}{\log(m/k)} \tag{10}$$

Since m/k can be considered constant, we can write

$$m \cdot l \approx m \cdot \log(N/m) \tag{11}$$

whereas $N \gg m$. Figure 6 shows a graph plot of Equation 11 for $N = 10^6$ (the black line). In practice we must use an integer number of levels (rounded up). This is shown as the grey line. You can see spikes where the number of levels changes.

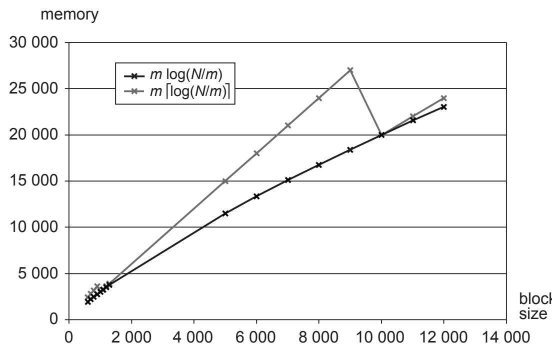

Figure 6: Graph plot of memory requirements.

It would seem that it is best to process data in very small blocks, but there is a drawback. When processing points in distinct blocks the result is an approximation of an ideal clustering. Of course the smaller the block, the worse the approximation. See [Guha et al. 2003] for details. According to our experiments, when varying block size, the clustering also varies somewhat but it still looks well. The major difference is that clustering in small blocks produces higher number of smaller clusters. Table 1 summarizes our experiments with the Lucy model [Stanford 2007], 10 072 906 vertices. The facility cost was set equal to the bounding box diagonal.

Table 1: The influence of block size on the clustering.

block size [points]	time [h:m]	memory [MB]	number of clusters at particular levels	
			level 1	level 2
1250	1:01	4.44	29 047	404
2500	1:58	3.97	28 400	366
5000	3:51	4.09	28 298	337
7500	5:43	5.27	28 270	336
10000	7:36	5.78	28 062	303

7.4 The Number of Iterations

[Charikar and Guha 1999] proved that $\mathcal{O}(m \log m)$ local search iterations are necessary for a constant factor approximation to the facility location. If we use large blocks, running time grows unpleasantly. We have made experiments with the number of iterations and it seems that it can be reduced significantly without major impact on results. Only about $0.1m$ iterations were necessary for uniformly distributed data. Data with obvious clusters required even less iterations.

You can see examples of clustering in Figure 7. The data set contains 1640 points. They were processed in a single block with facility cost equal to the bounding box diagonal. Black dots indicate points assigned to a different facility than to the closest one. It can be used as an approximate measure of error. But remember that it takes into account only the current set of facilities. No black dots mean optimal assignment to *currently open* facilities. With a different set of facilities, the clustering may be better.

Figure 7 a) shows the result after $m\lceil\log m\rceil = 6560$ iterations, clustering cost is 187. Figure 7 b) shows the result after $0.1m = 164$ iterations; clustering cost is 194.6 which is 4% more then a). Table 2 summarizes experiments with the Lucy model. The facility cost was set equal to the bounding box diagonal. You can compare time required for $m \log m$ and $0.1m$ iterations. If we use the reduced number of iterations, the clustering cost is slightly higher in each block. The table records statistics about this error so you can review the impact on clustering quality.

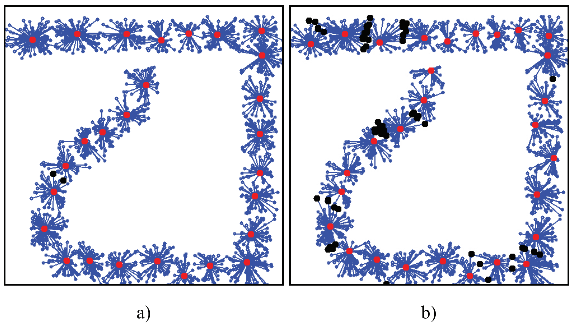

Figure 7: Clustering results a) after 6560 iterations, b) after 164 iterations.

8 Conclusion and future work

We have presented a modified data stream approach to solve the facility location problem on geometric data. We suggested an improvement to the facility location algorithm to limit the number of points inspected when computing the gain function. We proposed the facility weight normalization so that clustering works correctly at higher levels. We also performed experiments on clustering geometric data, described algorithm behaviour in various situations and discussed proper settings of particular parameters.

As a future work we would like to use this clustering method to create multiresolution geometric models where the user can select different levels of detail in various parts. It might be also interesting to add topological constraints so that points from different parts of model will not be joined into one cluster. Another interesting task is to cluster a triangular mesh so that clusters are consistent with the mesh topology. We could also take into account distribution of points in the data stream. Points arriving short one after another will be assigned to the same cluster, while points from different parts of the stream (even geometrically close together) will go into different clusters.

References

CHARIKAR, M., AND GUHA, S. 1999. Improved combinatorial algorithms for the facility location and k-median problems. In *IEEE Symposium on Foundations of Computer Science*, 378–388.

CHARIKAR, M., KHULLER, S., MOUNT, D. M., AND NARASIMHAN, G. 2001. Algorithms for facility location problems with outliers. In *Symposium on Discrete Algorithms*, 642–651.

CHARIKAR, M., GUHA, S., ÉVA TARDOS, AND SHMOYS, D. B. 2002. A constant-factor approximation algorithm for the k-median problem. *Journal of Computer System Sciences 65*, 1, 129–149.

CHARIKAR, M., O'CALLAGHAN, L., AND PANIGRAHY, R. 2003. Better streaming algorithms for clustering problems. In *Proc. of 35th ACM Symposium on Theory of Computing (STOC)*, 30–39.

CHUDAK, F. A., AND SHMOYS, D. B. 2004. Improved approximation algorithms for the uncapacitated facility location problem. *SIAM Journal on Comp. 33*, 1, 1–25.

FRAHLING, G., AND SOHLER, C. 2005. Coresets in dynamic geometric data streams. In *Proceedings of the 37th annual ACM symposium on Theory of computing (STOC)*, 209–217.

GUHA, S., AND KHULLER, S. 1998. Greedy strikes back: Improved facility location algorithms. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 649–657.

GUHA, S., RASTOGI, R., AND SHIM, K. 1998. CURE: An efficient clustering algorithm for large databases. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, 73–84.

GUHA, S., MISHRA, N., MOTWANI, R., AND O'CALLAGHAN, L. 2000. Clustering data streams. In *IEEE Symposium on Foundations of Computer Science*, 359–366.

GUHA, S., MEYERSON, A., MISHRA, N., MOTWANI, R., AND O'CALLAGHAN, L. 2003. Clustering data streams: Theory and practice. *IEEE Transactions on Knowledge and Data Engineering 15*, 3, 515–528.

ISENBURG, M., AND GUMHOLD, S. 2003. Out-of-core compression for gigantic polygon meshes. In *SIGGRAPH'03 Conference Proceedings*, 935–942.

ISENBURG, M., AND LINDSTROM, P. 2005. Streaming meshes. In *Proceedings of Visualization'05*, 231–238.

ISENBURG, M., LINDSTROM, P., AND SNOEYINK, J. 2005. Streaming compression of triangle meshes. In *Proceedings of the 3rd Eurographics symposium on Geometry processing (SGP)*, 111.

ISENBURG, M., LINDSTROM, P., GUMHOLD, S., AND SHEWCHUK, J. 2006. Streaming compression of tetrahedral volume meshes. In *Graphics Interface*, 115–121.

ISENBURG, M., LIU, Y., SHEWCHUK, J., AND SNOEYINK, J. 2006. Streaming computation of delaunay triangulations. *ACM Trans. Graph. 25*, 3, 1049–1056.

JAIN, K., AND VAZIRANI, V. V. 1999. Primal-dual approximation algorithms for metric facility location and k-median problems. In *IEEE Symposium on Foundations of Computer Science*, 2–13.

JAIN, A. K., MURTY, M. N., AND FLYNN, P. J. 1999. Data clustering: A review. *ACM Computing Surveys 31*, 3, 264–323.

KAUFMAN, L., AND ROUSSEEUW, P. J. 1990. *Finding groups in data: An introduction to cluster analysis.* John Wiley, New York.

KORUPOLU, M. R., PLAXTON, C. G., AND RAJARAMAN, R. 1998. Analysis of a local search heuristic for facility location problems. In *9th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 1–10.

LINDSTROM, P. 2000. Out-of-core simplification of large polygonal models. In *Siggraph 20000, Computer Graphics Proceedings*, 259–262.

MAHDIAN, M., MARKAKIS, E., SABERI, A., AND VAZIRANI, V. V. 2001. A greedy facility location algorithm analyzed using

Table 2: Experiments on the number of iterations.

block size [points]	time for $m \log m$ [h:m:s]	time for $0.1m$ [h:m:s]	percent of time used for $0.1m$	min. error	avg. error	max. error
1250	1:01:24	0:02:41	4.4%	0	1.5%	14.8%
2500	1:58:00	0:04:29	3.8%	0.2%	1.6%	5.0%
5000	3:51:01	0:07:45	3.4%	0.5%	1.7%	4.9%
7500	5:43:21	0:11:09	3.3%	0.5%	1.7%	3.5%
10000	7:36:36	0:14:32	3.2%	0.8%	1.6%	3.2%

dual fitting. In *4th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, 127–137.

MEYERSON, A. 2001. Online facility location. In *Proceedings of the 42nd IEEE symposium on Foundations of Computer Science (FOCS)*, 426.

MUTHUKRISHNAN, S. 2003. Data streams: Algorithms and applications. In *Proceedings of the 14th annual ACM-SIAM symposium on discrete algorithms*.

NG, R. T., AND HAN, J. 1994. Efficient and effective clustering methods for spatial data mining. In *20th Intl. Conference on Very Large Data Bases*, 144–155.

O'CALLAGHAN, L., MISHRA, N., MEYERSON, A., GUHA, S., AND MOTWANI, R. 2002. Streaming-data algorithms for high-quality clustering. In *18th International Conference on Data Engineering (ICDE)*, 685.

PAJAROLA, R. 2005. Stream-processing points. In *Proceedings IEEE Visualization*, 239–246.

ROSSIGNAC, J. R., AND BORREL, P. 1993. Multi-resolution 3D approximations for rendering complex scenes. In *Geometric Modeling in Comp. Graphics*, 455–465.

SHARIFZADEH, M., AND SHAHABI, C. 2004. Approximate voronoi cell computation on geometric data streams. Tech. Rep. 04-835, University of Southern California, Computer Science Department.

SHMOYS, D. B. 2000. Approximation algorithms for facility location problems. In *Proceedings of International Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX)*, 27–33.

STANFORD, 2007. Stanford 3D scanning repository. http://graphics.stanford.edu/data/3Dscanrep/.

ZHANG, T., RAMAKRISHNAN, R., AND LIVNY, M. 1996. BIRCH: An efficient data clustering method for very large databases. In *ACM SIGMOD International Conference on Management of Data*, 103–114.