A Divisive Hierarchical Clustering Algorithm To Reduce Dissection Effect Using Greedy Heuristics

Sadman Sadeed Omee and Md. Saidur Rahman

Department of Computer Science and Engineering,
Bangladesh University of Engineering and Technology,
Dhaka, Bangladesh
omee.sadman@gmail.com
saidurrahman@cse.buet.ac.bd

Abstract. We consider the problem of *dissection effect* here. The problem is caused when objects naturally belonging to the same cluster are assigned to separate clusters. As we want data clusters to be compact, it sometimes results in unnecessary splitting of a natural cluster. For n data points and at most k clusters allowed, we present an $O(nk \log n)$ algorithm for reducing this problem and produce good clusters on average. Our algorithm is a divisive hierarchical clustering algorithm and it follows a greedy heuristic. We test our algorithm with different benchmark data sets. We compare our results to the very well known k-means clustering. We discuss the drawbacks of k-means clustering by finding cases where it works poorly. We try to overcome these cases in our algorithm. We present and analyze the results graphically.

Keywords: Dissection Effect · k-means · Centroid · Diameter · Unsupervised Learning · Divisive Hierarchical Clustering · Greedy heuristic.

1 Introduction

Clustering is arguably the most important section of Unsupervised Learning where data are partitioned into sensible groups. The goal of clustering is to divide an unlabelled data set into separate groups according to their similarity, to find underlying structure in data and to organize and summarize it through cluster prototypes [13] Clustering algorithms are developed to find clusters among data by following some objective function.

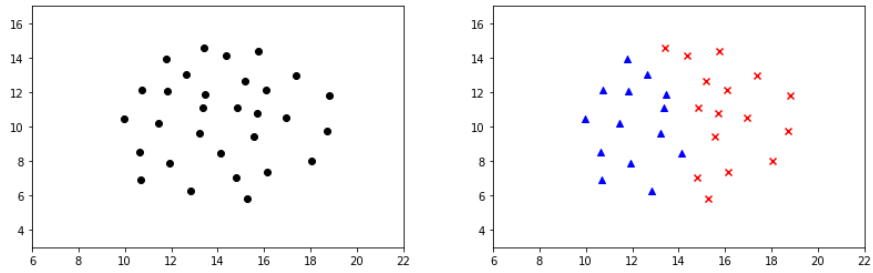
Finding proper and appropriate clusters is considered a very complex problem. In fact, clustering is regarded as a harder and challenging problem than classifications [13]. In spite of its being a difficult problem, clustering algorithms are used heavily in various sectors on a regular basis including pattern recognition and information retrieval, image segmentation, business and marketing, genetics, medical imaging etc.

Various clustering algorithms have been developed over the years because of its significance. They are of different types and their objective function varies.

They can also vary on what heuristic they follow. As huge amount of data are being produced every second nowadays, the importance of efficient data clustering algorithm has increased rapidly for the purpose of mining these huge amount of data. Moreover, the data itself can be of several types and dimensions. Each and every clustering algorithm are designed to fulfill specific objective and handle data of specific type and dimensions and the success of these algorithm on how effectively they can handle the data.

1.1 Dissection Effect

We emphasize on the problem of *dissection effect* here. This problem has been mentioned in several researches before (see [12, 6, 15, 7]). The problem arises when objects very similar to one another are assigned to separate clusters instead of assigning them to the same cluster. When we desire clustering algorithms to find compact clusters, so that diameter or radius of one cluster does not become much larger than others, they usually tend to find clusters of almost the same size. This sometimes results in finding clusters where the intra-cluster distance is big or splitting of a natural cluster which should not have been split. Fig. 1 shows an examples of dissection effect where all the points belonged to the same cluster but split and assigned to two separate clusters and fig. 2 shows an example of dissection effect where it occur from a point naturally belonged to one cluster but assigned to a different one by the clustering algorithm.



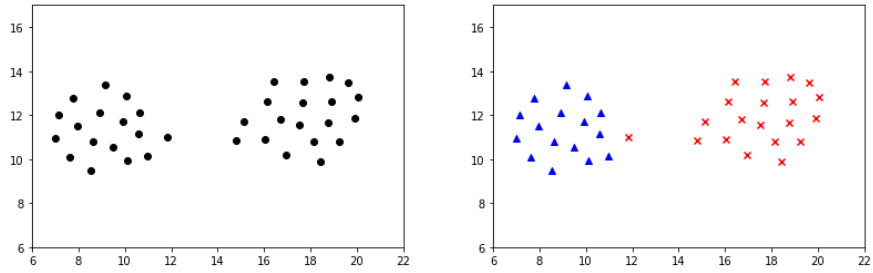
(i) A point set (ii) A clustering of the point set

Fig. 1: An example of dissection effect where it arises from oversplitting the data.

Dissection Effect depends on the objective function selected to perform the clustering. Every objective function produces different results and the dissection effect varies with them. In the next subsection, we describe objective functions used before to reduce dissection effect.

1.2 Related Works

The dissection effect generally occurs when the objective function is to minimize the maximum radii or diameters of the clusters. To reduce this problem, this



(i) A point set (ii) A clustering of the point set

Fig. 2: An example of dissection effect where it arises from assigning a point to the wrong cluster.

objective function was modified and a new objective function was proposed. Minimizing the sum of radii or minimizing the sum of diameters of all the clusters proved to be a more effective objective function than minimizing the maximum radius/diameter for reducing dissection effect [12, 15].

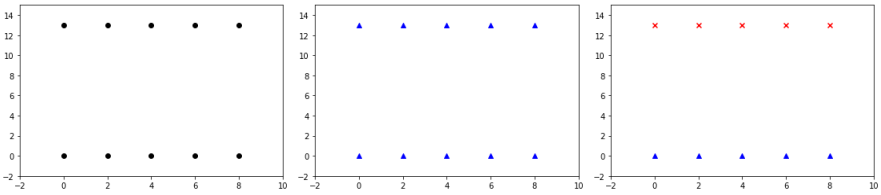
Given n objects $p_1, p_2, p_3, \ldots, p_n$ and an integer k, clustering algorithms with minimizing the sum of radii/diameters as objective function, divide the objects of into at most k clusters so that the sum of radii/diameters of these clusters is minimized. These are known as the *Minimum Sum of Radii* and *Minimum Sum of Diameters* problem, respectively. Minimum Sum of Diameters and Minimum Sum of Radii problem has been well researched where the motivation was to reduce dissection effect(see [8, 5, 4, 2]). For k clusters, Doddi *et al.* [8] devised an approximation algorithm that comes up with a solution which finds at most $10k$ clusters such that the solution is a logarithmic factor of the optimal solution. For k fixed clusters, they also presented a $2-$approximation algorithm. For any $\epsilon > 0$, they showed that it is NP-Hard to calculate any approximation factor $2 - \epsilon$ for minimizing the sum of diameters.

Chaikar and Panigrahy [5] improved these results and devised a primal-dual based algorithm which achieves a constant factor approximation using at most k clusters. They obtained a $(3.504 + \epsilon)-$approximation for Minimum Sum of Radii problem.

Gibson *et al.* [11] presented an exact algorithm in time that had $n^{O(\log_n \log \Delta)}$ complexity for Minimum Sum of Radii problem, where Δ is the ratio of the maximum to minimum inter-point distance. It is a Quasi-Polynomial Time Approximation Scheme (QPTAS) for Minimum Sum of Radii problem.

Behsaz and Salavatipour [2] they gave an exact algorithm for the Minimum Sum of Radii problem when singleton clusters (clusters with a single point) are not allowed. They devised a Polynomial Time Approximation Scheme (PTAS) for the Minimum Sum of Diameters problem on the plane with Euclidean distances. For Minimum Sum of Diameters problem with constant k, they present a polynomial time exact algorithm.

Although minimizing the sum of radii/diameters is a good objective function for reducing dissection effect, it can not reduce it in every case. Fig. 3 shows a case where this objective function does not produce output with minimum dissection effect for $k = 2$. Again, the running time complexity of the algorithms developed for this purpose is very high which is another problem. So we need to find a faster way for reducing this problem effectively in most cases.



(i) A point set

(ii) Minimum sum of diameters clustering of the point set

(iii) Clustering of the point set with minimum dissection effect

Fig. 3: A case where minimum sum of diameters objective does not produce output with minimum dissection effect for $k = 2$.

1.3 Our Contributions

In this paper, we study the problem on point sets. Our algorithm can be extended for higher dimensions but in this paper we focus on 2-dimensional points. We present a $O(nk \log n)$ algorithm for reducing dissection effect, where n is the number of point in the data set and k is the maximum number of clusters allowed. The algorithm uses a greedy heuristic to find clusters. It is a hierarchical clustering algorithm that follows top-down approach i.e., a divisive hierarchical clustering algorithm. Our algorithm works better than the previous algorithms in many cases. We test our algorithm on various data sets. We also test these data sets using different distance metrics, mostly Euclidean, Manhattan and Chebyshev Distance.

Finding good clusters is always the objective of any clustering algorithm. So we also wanted to test our algorithm against other well known clustering algorithms. We chose k-Means algorithm [14] for this purpose. Berkhin [3] stated in his survey that "The k-means algorithm is by far the most popular clustering tool used in scientific and industrial applications". Lloyd [14] proposed a local search solution to this problem which is the most popular version of the k-Means algorithm. Though k-Means is the the most used and popular algorithm, its running time is superpolynomial i.e., exponential in worst case [1, 17]. We test our algorithm against this widely known algorithm and overcome some cases where the k-Means algorithm fails to generate good clusters. We also compare the accuracy of our algorithm and k-means clustering for some benchmark datasets and present results in tables and graphs.

2 Preliminaries

Finding good clusters is a very complex problem. A good clustering algorithm should find clusters should find clusters so that intra-cluster cluster similarity is high as possible and inter-cluster similarity is as low as possible. Anil K. Jain [13] stated in his paper that "there is no best clustering algorithm". Every clustering algorithm tries to follow these properties by maintaining an objective function.

The *radius* of a cluster is the maximum distance between the centroid and all the points cluster. On the other hand, the *diameter* of a cluster is the maximum distance between any two points of the cluster. The radius and diameter of a cluster are not related directly, but they tend to be proportional.

Our algorithm is a hierarchical algorithm and a divisive one. A divisive hierarchical clustering algorithm starts by putting all the points in a single cluster and then recursively divides the cluster into smaller clusters [13]. Divisive algorithms usually find more correct solution than agglomerative algorithms. Agglomerative algorithms (bottom-up approach) does not have idea about the whole data pattern initially, it proceeds by finding patterns locally. But divisive algorithms (top-down approach) does have idea about the whole data pattern initially and so it makes better choices during clustering.

The objective function of k-Means algorithm is to minimize intra-cluster variance i.e., sum of squared error (SSE). For every data point p and centroid c, the sum squared error can be defined as follows where p_j^i is the j^{th} data point assigned to i^{th} cluster.

$$SSE = \sum_{i=1}^{k} \sum_{j=1}^{n} \left|\left| p_j^i - c_i \right|\right|^2$$

The similarity measure can vary among clustering algorithms. We use distance between two points as similarity measure for our algorithm in this paper. k-Means algorithm also uses the same similarity measure. We mostly use three types of Minkowski Distances as distance metrics for this purpose - Euclidean Distance, Manhattan Distance and Chebyshev Distance. For two points $X = \{x_1, x_2, \ldots, x_n\}$ and $Y = \{y_1, y_2, \ldots, y_n\}$, the Minkowski Distance of order p between two points is defined as,

$$Minkowski(X, Y) = \left(\sum_{i=1}^{n} \left|\left| x_i - y_i \right|\right|^p \right)^{\frac{1}{p}}$$

$p = 1, 2$ and ∞ is equivalent to Manhattan Distance, Euclidean Distance and Chebyshev Distance, respectively.

Notice that, we are using the term point set instead of a point array. In a point array, the same point can be multiple times as there can be more than one point in a specific coordinate. But a point set can not have an element more than once. Besides, there is no need to handle multiple points in a specific coordinate as we consider all the points in a specific coordinate as a single point lying in that coordinate. The reason behind this is that closer points tend to be in the

same cluster and so as no matter how many points are on a specific location, they all tend to be in the same cluster. That is why we do not need to bother about multiple points in the same coordinate during clustering. So putting all the input points in a point set is a better option.

3 The Algorithm

In this section, we describe our algorithm. Let $S = \{p_1, p_2, p_3, \ldots, p_n\}$ be a point set of n points and k be an integer. The goal is to divide the point set into at most k clusters so that similar points are assigned to same clusters and dissimilar points are assigned to different clusters. We divide our algorithm into four major steps. They are the following −

1. Initially dividing the point set S into two clusters C_1 and C_2 in such a way that number of points in both the clusters are as close as possible. We name it the *"Initial Divide"* step.
2. Finding points those are actually nearer to the centroid of S than the centroid of the cluster they are assigned to. We remove those points from their corresponding cluster and store it in a temporary cluster C_{temp}. We name it the *"Temporary Cluster Creation"* step.
3. Applying a greedy heuristic and merging the temporary cluster (C_{temp}) found in step 2 with one of the clusters between C_1 and C_2. We use the greedy heuristic to determine which cluster between C_1 and C_2 we need to merge C_{temp} with. We call it the *"Merge by Greedy Heuristic"* step.
4. Finally we remove all the points of C_1 those are nearer to the centroid of C_2 than that of C_1 and assign them to C_2. Then we do the same for C_2, that is, we remove all the points of C_2 those are nearer to the centroid of C_1 than that of C_2 and assign them to C_1. We call it the *"Filtering"* step.

These four steps creates two clusters C_1 and C_2 from point set S. Our objective is to get $\leqslant k$ clusters. According to the rule of a divisive hierarchical clustering algorithm, we need to repeat the above mentioned four steps in either C_1 or C_2 and so on to divide into more clusters. For k clusters, we need to use these steps $k - 1$ times. Our algorithm follows another greedy heuristic for deciding to pick which cluster to divide in the next iteration.

We also need to track the divide of the point set S that has the least dissection effect. Suppose we have $k = 15$, but if we divide it more than 12 clusters, the dissection effect increases. And also if we divide it less than 12 clusters, the dissection effect increases. So we need to save the *state* of clustering of each $k = 1, 2, 3, \ldots, 15$. The state for each k here represents the clustering of the points for exactly k clusters by our algorithm. $k = 1$ means that there is no clustering needed for S, as dividing S will only increase the dissection effect. We then output the state that has the minimum dissection effect.

In subsection sections 3.1 to 3.4 we discuss the four major steps in details.

3.1 Initial Divide

Our algorithm first divides S into two clusters C_1 and C_2. Our algorithm starts with finding the two farthest points f_1 and f_2 of the point set S. After finding the two farthest point f_1 and f_2 of the points set S, we assign f_1 to C_1 and f_2 to C_2 and remove f_1 and f_2 from S. Then we compute nearest points of f_1 and f_2 in S and assign them to C_1 and C_2, respectively. Then we also remove them from S. We repeat the process of finding nearest points of the last assigned points of C_1 and C_2 and assigning them into clusters and removing them from the point set until S is empty. The algorithm is presented below.

Algorithm 1: $InitialDivide(S)$

Input: The point set S
Output: Two clusters C_1 and C_2

$C_1, C_2 \leftarrow \phi$
$f_1, f_2 \leftarrow$ farthest point pair of S
$S \leftarrow S \setminus \{f_1, f_2\}$
$C_1 \leftarrow C_1 \cup \{f_1\}$
$C_2 \leftarrow C_2 \cup \{f_2\}$
$q_1 \leftarrow f_1$
$q_2 \leftarrow f_2$
while S *is not empty* **do**
    $x_1 \leftarrow$ nearest point of q_1 of S
    $S \leftarrow S \setminus \{x_1\}$
    $C_1 \leftarrow C_1 \cup \{q_1\}$
    $q_1 \leftarrow x_1$
    if S *is not empty* **then**
        $x_2 \leftarrow$ nearest point of q_2 of S
        $S \leftarrow S \setminus \{x_2\}$
        $C_2 \leftarrow C_2 \cup \{q_2\}$
        $q_2 \leftarrow x_2$
    end
end
return C_1, C_2

Computing farthest point pair of a $2d$ point set takes total $O(n \log n)$ complexity by rotating calipers method [16]. This method first requires computing the convex hull of the point set which takes $O(n \log n)$ time. Then it searches for the two farthest points in $O(n)$ time with the rotating calipers. So in total it takes $O(n \log n)$ time. Nearest point query of a specific point within a point set takes $O(\log n)$ time in the expected case by using kd-trees [10].

3.2 Temporary Cluster Creation

In this step, we create a temporary cluster. We name it C_{temp}. This cluster is needed because some points lies closer to the mean of the whole point set

rather than the centroid of the cluster it is assigned to in the previous step. After removing these points from their corresponding clusters, we assign them temporarily to C_{temp}. If the points of C_{temp} comes from only one of the clusters between the two found in the previous step, then this step is not very significant. But if the points come from both the clusters, then this step is really vital. The algorithm is presented below.

Algorithm 2: *CreateTemporaryCluster*(C_1, C_2)

Input: Two clusters C_1 and C_2
Output: Updated Clusters C_1, C_2 and a temporary cluster C_{temp}

$c_1 \leftarrow$ centroid of C_1
$c_2 \leftarrow$ centroid of C_2
$cent \leftarrow$ mid point of c_1 and c_2 ▷ centroid of C_1 and C_2 combined
$C_{temp} \leftarrow \phi$
for $i \leftarrow C_1.begin$ *to* $C_1.end$ **do**
  $x \leftarrow i^{th}$ point of C_1
  $dist\_x\_c_1 \leftarrow$ distance between x and c_1
  $dist\_x\_cent \leftarrow$ distance between x and $cent$
  if $dist\_x\_cent \leqslant dist\_x\_c_1$ **then**
    $C_{temp} \leftarrow C_{temp} \cup \{x\}$
    $C_1 \leftarrow C_1 \setminus \{x\}$
  end
end
for $i \leftarrow C_1.begin$ *to* $C_2.end$ **do**
  $x \leftarrow i^{th}$ point of C_2
  $dist\_x\_c_2 \leftarrow$ distance between x and c_2
  $dist\_x\_cent \leftarrow$ distance between x and $cent$
  if $dist\_x\_cent \leqslant dist\_x\_c_2$ **then**
    $C_{temp} \leftarrow C_{temp} \cup \{x\}$
    $C_2 \leftarrow C_2 \setminus \{x\}$
  end
end
return C_1, C_2, C_{temp}

We denote the distance between two point x_1 and x_2 by x_1x_2. Let, C be a cluster and c be its centroid. Suppose, f is the farthest from from c within all the points of C. Then we define the *range* of cluster C as the circular area taking cf as the radius. Any point that is inside this circle is within the range of cluster C. The intuition of this step comes from the following Lemma -

Lemma 1. *Let, $p_1 \in C_1$ and $p_2 \in C_2$ and c_1 and c_2 be their centroid, respectively. Suppose, they are assigned to C_{temp} by algorithm 2. Then for some point f_1 collinear with p_1 and c_1 and some point f_2 collinear with p_2 and c_2 where $p_1c_1 = f_1c_1$ and $p_2c_2 = f_2c_2$, it can be stated that, p_1p_2 is at least smaller than either p_1f_1 or p_2f_2.*

Proof. We denote the centroid of C_1 and C_2 combined by c. We choose $f_1(f_2)$ in such a way that $c_1(c_2)$ is equidistant from $p_1(p_2)$ and $f_1(f_2)$ and $p_1(p_2)$, $c_1(c_2)$, $f_1(f_2)$ are collinear. The reason is described below -

We consider a circle with p_1c_1 as radius. If p_1 is the farthest point of C_1 from c_1, then the circle will cover all the points of C_1. Else, at least point of C_1 will be outside of the circle. So in either case, f_1 is in the range of C_1. So we can consider a point at f_1 and if that point belonged the actual point set, it would have been assigned to C_1 as it is well inside the range of C_1. Same goes for f_2 being in the range of C_2.

Now, as, p_1 and p_2 are assigned to C_{temp}, so $p_1c < p_1c_1$ and $p_2c < p_2c_2$. Combining these,

$$p_1c + p_2c < p_1c_1 + p_2c_2 \tag{1}$$

We consider 3 cases.
Case 1: $p_1c > p_2c$
Equation 1 becomes,

$$\begin{aligned} p_1c + p_2c &< p_1c_1 + p_1c_1 \\ &= p_1c_1 + f_1c_1 \\ &= p_1f_1 \\ \therefore p_1c + p_2c &< p_1f_1 \end{aligned} \tag{2}$$

Now, $p_1p_2 \leqslant p_1c + p_2c$, because of triangle inequality. Then equation 2 becomes,

$$\begin{aligned} p_1p_2 \leqslant p_1c + p_2c &< p_1f_1 \\ \therefore p_1p_2 &< p_1f_1 \end{aligned} \tag{3}$$

Case 2: $p_1c < p_2c$
Similar to case 1 we prove it for f_2 this time. Finally we get,

$$p_1p_2 < p_2f_2 \tag{4}$$

Case 3: $p_1c = p_2c$
Here both case 1 and 2 true. So we get,

$$p_1p_2 < p_1f_1 \textit{ and } p_1p_2 < p_2f_2 \tag{5}$$

Combining equation 3, 4 and 5, we can say that, p_1p_2 is at least smaller than either p_1f_1 or p_2f_2.

Lemma 1 indicates that despite having a larger distance, if (p_1, f_1) pair or (p_2, f_2) pair deserve to be in the same cluster, then (p_1, p_2) pair also deserves to be in the same cluster even more. That is why, all the points assigned to C_{temp} are merged with either C_1 or C_2 in the next step so that they remain in the same cluster.

3.3 Merge by Greedy Heuristic

We get three clusters C_1, C_2 and C_{temp} from the previous step. In this step, we merge C_{temp} with either C_1 or C_2 using a greedy heuristic.

First we describe the greedy heuristic. We choose the ratio of the diameter of a cluster and the number of points of that cluster as the greedy heuristic for choosing between C_1 and C_2 for merging. This ratio gives the idea of the spacing of the points in every direction within the cluster. We present a helper function for calculating this ratio below.

Algorithm 3: $CalculateRatio(C)$

Input: A cluster C
Output: Ratio of diameter and number of points of C
$d \leftarrow$ diameter of C
$ratio \leftarrow \frac{d}{C.length}$
return $ratio$

As the final cluster pairs after this step would either be $(C_1 + C_{temp}, C_2)$ or $(C_1, C_2 + C_{temp})$, so we calculate the ratio value for all these four clusters and always output that pair that has lower ratio value sum. This is because, lower ratio value means more compact clusters which is one of the most important properties of a good cluster. The algorithm is presented below.

Algorithm 4: $MergeByGreedyHeuristic(C_1, C_2, C_{temp})$

Input: Three clusters C_1, C_2 and C_{temp}
Output: Updated Clusters C_1 and C_2
$C_1^* \leftarrow C_1 \cup C_{temp}$
$C_2^* \leftarrow C_2 \cup C_{temp}$
$ratio_1 \leftarrow CalculateRatio(C_1^*) + CalculateRatio(C_2)$
$ratio_2 \leftarrow CalculateRatio(C_1) + CalculateRatio(C_2^*)$
if $ratio_1 \leqslant ratio_2$ **then**
  $C_1 \leftarrow C_1^*$
else
  $C_2 \leftarrow C_2^*$
end
delete C_1^*, C_2^* and C_{temp}
return C_1, C_2

3.4 Filtering

We got the updated C_1 and C_2 in the last step. The filtering step is plain and simple. It takes all the points of C_1 those are nearer to the centroid of C_2 than that of C_1 and assigns them to C_2. Similarly, it takes all the points of C_2 those are nearer to the centroid of C_1 than that of C_2 and assigns them to C_1. The algorithm is presented below.

Algorithm 5: $Filtering(C_1, C_2)$

Input: Two clusters C_1 and C_2
Output: Updated Clusters C_1 and C_2

$c_1 \leftarrow$ centroid of C_1
$c_2 \leftarrow$ centroid of C_2
for $i \leftarrow C_1.begin$ *to* $C_1.end$ **do**
  $x \leftarrow i^{th}$ point of C_1
  $dist\_c_1 \leftarrow$ distance between x and c_1
  $dist\_c_2 \leftarrow$ distance between x and c_2
  if $dist\_c_1 > dist\_c_2$ **then**
    $C_1 \leftarrow C_1 \setminus \{x\}$
    $C_2 \leftarrow C_2 \cup \{x\}$
  end
end
for $i \leftarrow C_2.begin$ *to* $C_2.end$ **do**
  $x \leftarrow i^{th}$ point of C_2
  $dist\_c_1 \leftarrow$ distance between x and c_1
  $dist\_c_2 \leftarrow$ distance between x and c_2
  if $dist\_c_1 < dist\_c_2$ **then**
    $C_2 \leftarrow C_2 \setminus \{x\}$
    $C_1 \leftarrow C_1 \cup \{x\}$
  end
end
return C_1, C_2

3.5 Putting It All Together

Here we combine all the steps and complete our divisive hierarchical algorithm. As we said earlier, algorithm algorithms 1 to 5 divides a point set into two clusters. We then present a helper function below combining all these steps which is a part of every iteration of our divisive hierarchical clustering algorithm.

Algorithm 6: $CreateTwoClusters(C)$

Input: A cluster C
Output: Clustering of C to create two new clusters C_1 and C_2

$C_1, C_2 \leftarrow InititalDivide(C)$
$C_1, C_2, C_{temp} \leftarrow CreateTemporaryCluster(C_1, C_2)$
$C_1, C_2 \leftarrow MergeByGreedyHeuristic(C_1, C_2, C_{temp})$
$C_1, C_2 \leftarrow Filtering(C_1, C_2)$
return C_1, C_2

We discussed earlier that we apply another greedy heuristic to choose a cluster among remaining clusters to divide in the next iteration. For this, we calculate the ratio value(diameter and number of points ratio) of each cluster. We then create a state with the cluster along with its ratio value and store it in a set.

Then we extract the cluster in that set that has the maximum ratio value(greedy heuristic), because larger ratio value means less compact cluster and so this needs to be divided next. So basically this greedy heuristic is kind of opposite of the previous one.

Also we need to keep track of the clustering that has the least dissection effect. So we calculate the average of the ratio values according to the distribution in each iteration. We define a variable *minAvgRatio* that keeps track of the minimum average ratio value. Also we store the clustering states till that iteration if the average ratio value in that iteration is minimum(assigned to *minAvgratio*). We finally output the clustering that has the minimum average ratio value. The complete divisive hierarchical clustering algorithm is presented below.

Algorithm 7: $DHClustering(S)$

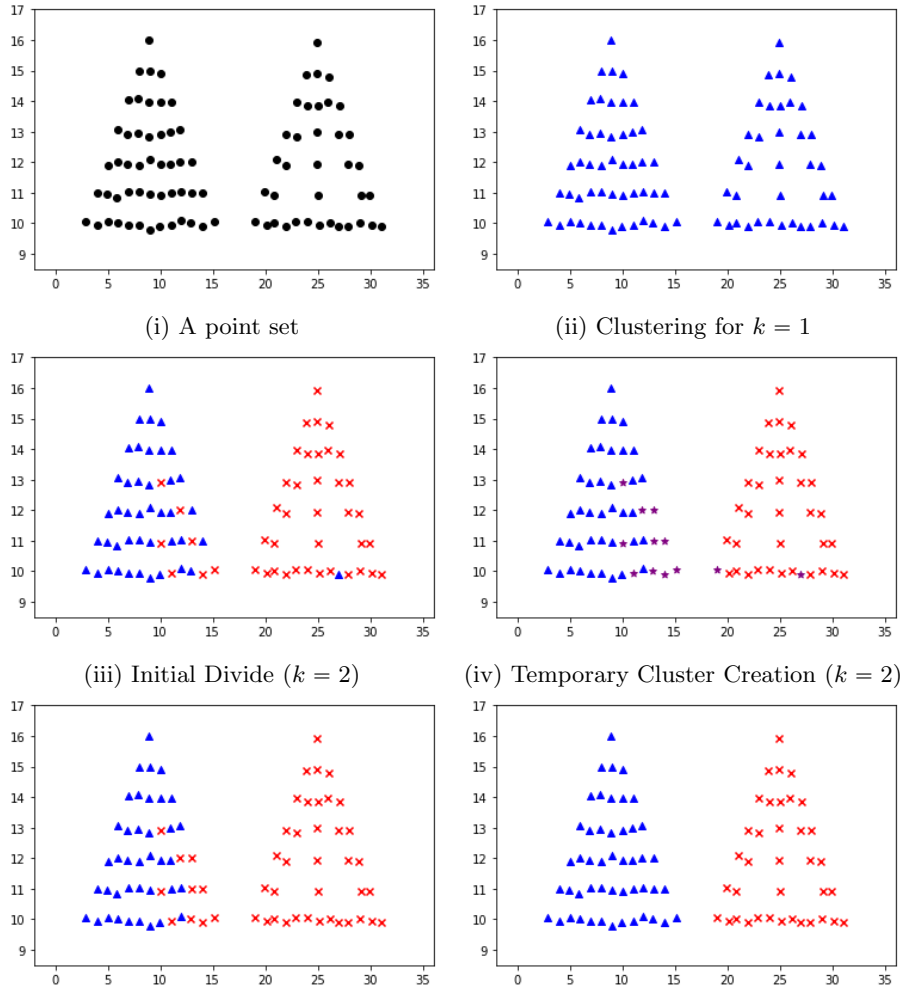
Input: The point set S and an integer k
Output: Less or equal k clusters of S

$C \leftarrow \phi$
$ratio \leftarrow CalculateRatio(S)$ ▷ *ratio* value of S
create a state c with S and *ratio*
insert c into C
$C_{state} \leftarrow C$
$minAvgRatio \leftarrow ratio$
for $i \leftarrow 1$ *to* $k-1$ **do**
  $state \leftarrow$ extract the state with the maximum *ratio* value from C
  $cluster \leftarrow$ extract the cluster from *state*
  $C_1, C_2 \leftarrow CreateTwoClusters(cluster)$
  $ratio_1 \leftarrow CalculateRatio(C_1)$ ▷ *ratio* value of C_1
  $ratio_2 \leftarrow CalculateRatio(C_2)$ ▷ *ratio* value of C_2
  create a state c_1 with C_1 and its *ratio* value
  insert c_1 into C
  create a state c_2 with C_2 and its *ratio* value
  insert c_2 into C
  $avgRatio \leftarrow$ calculate the average of *ratio* values of all clusters of C
  if $avgRatio < minAvgRatio$ **then**
    $C_{state} \leftarrow C$
    $minAvgRatio \leftarrow avgRatio$
  end
end
delete C
$clusters \leftarrow$ extract the clusters from C_{state}
return *clusters*

4 Results and Analysis

Fig. 4 shows step by step simulation of algorithm 7 for $k = 2$ for a point set using euclidean distance. For $k = 1$, the output of the clustering is just putting

the whole point set in one cluster. For $k = 2$, this cluster is then divided into two clusters. The points of the temporary cluster are marked by purple '*'. The final output is the output for $k = 2$ considering average ratio value in both cases.



(i) A point set

(ii) Clustering for $k = 1$

(iii) Initial Divide ($k = 2$)

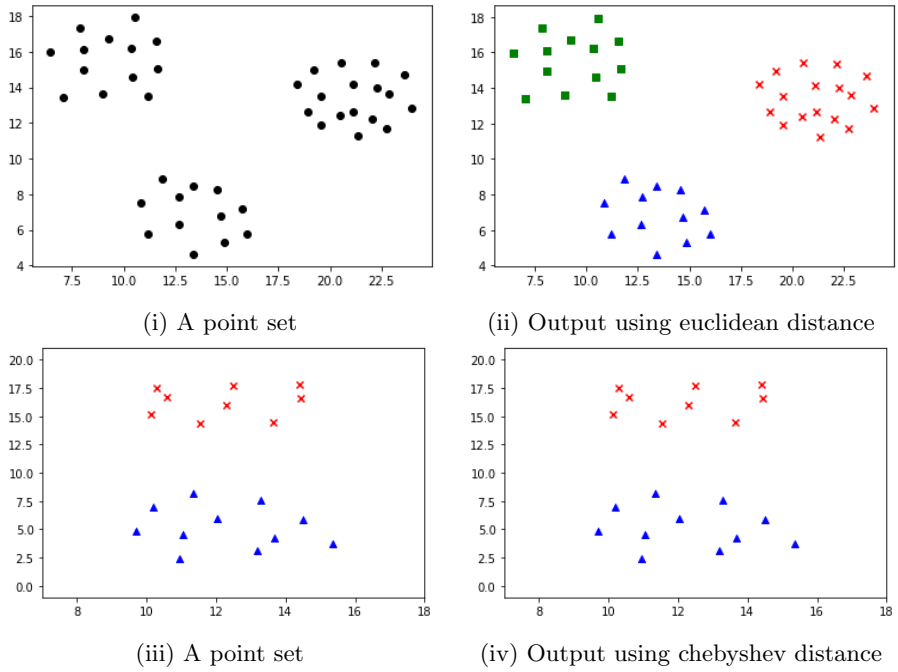
(iv) Temporary Cluster Creation ($k = 2$)

(v) Merge by Greedy Heuristic ($k = 2$)

(vi) Filtering ($k = 2$) and final output

Fig. 4: Step by step simulation of algorithm 7 for a point set using euclidean distance

Sample outputs of algorithm 7 for a point set for $k = 3$ using manhattan and chebyshev distance, respectively is shown in fig. 5.



(i) A point set

(ii) Output using euclidean distance

(iii) A point set

(iv) Output using chebyshev distance

Fig. 5: Sample output of algorithm 7 for $k = 3$ using manhattan and chebyshev distances

Now, we compare our algorithm with k-means clustering. We start by comparing their running time. For n $2d$ points and at most k clusters allowed, our algorithm has $O(nk \log n)$ complexity.

Lemma 2. *Algorithm 7 has $O(nk \log n)$ complexity.*

Proof. The farthest point pair calculation of a point set takes $O(n \log n)$ complexity. The nearest neighbor calculation of a specific point of a point set takes amortized $O(\log n)$ complexity. So the total running time of the 'initial divide' steps is $O(n \log n)$. The calculation of the centroid of point set and the calculation of distance between two points both take $O(n)$ time. So the 'temporary cluster creation' step has $O(n)$ complexity. The 'merge by greedy heuristic' step takes $O(n \log n)$ time as the most costly calculation of this step is to find diameter by number of points ratio which takes $O(n \log n)$ time. Finally, the 'filtering' step takes $O(n)$ time. Combining all these steps, we have algorithm 6 and so it has a complexity of $O(n \log n)$.

Now, algorithm 6 is a part of each iteration of our main algorithm. The iteration goes over for $k - 1$ times. The other parts of each iteration are finding ratio values of each cluster and finding the average ratio value for each iteration. So the calculations of each iteration has a complexity of $O(n \log n)$. So, consid-

ering $k-1$ iterations, finally we can say that, algorithm 7 has a complexity of $O(nk \log n)$.

The converge of k-means clustering is highly researched and it is known to have exponential running time in worst case [1, 17]. Also the accuracy of the k-means output is depended on the initialization because the iterations of k-means clustering can get stuck in a local minimum for bad initialization. Also, the quality of k-means clustering depends on the value of k. It is shown in [9] that, the success of k-means clustering has an inverse linear dependency on k. It is also shown in [9] that, k-means performs poorly when cluster sizes have a strong unbalance.

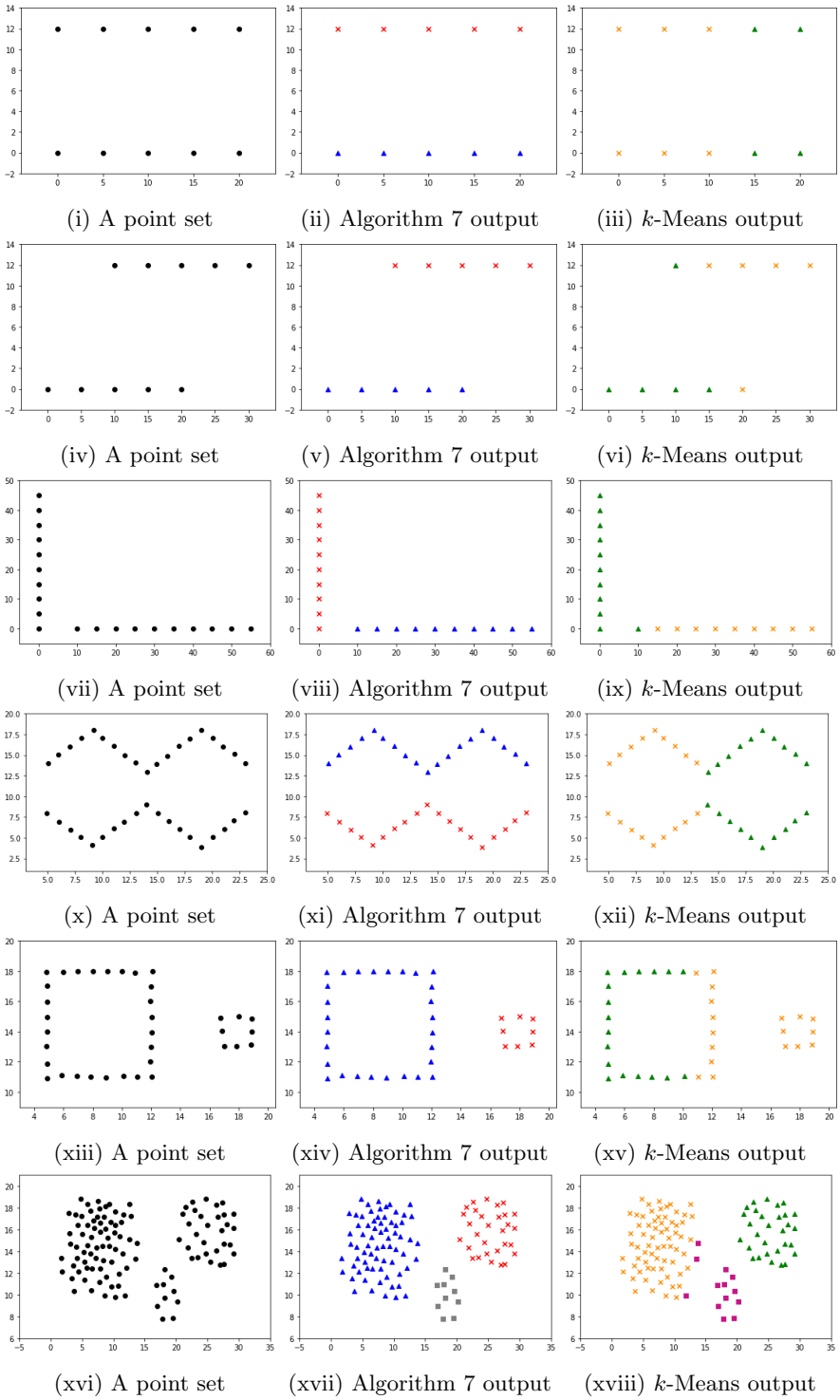
For this paper, for producing k-means output, we run k-means clustering 30 times with different centroid seeds. Then we find the run with the minimum sum squared error and choose it as the final output.

Fig. 6 shows some cases where k-means clustering fails to generate good clusters but our algorithm succeeds to find the proper clusters. The comparison of results of these cases are shown in Table 1. The problem generates because it tries to minimize the intra-cluster variance.

Now, we compare our algorithm with k-means clustering for some benchmark datasets. We select 8 datasets for this purpose - Aggregation, Flame, Unbalance, S1, A-Sets(A1, A2, A3) and Birch2. Details about these datasets can be found in [9].

We run our algorithm using two different metrics(euclidean and manhattan) and k-means clustering on these datasets. Table 2 shows the results and comparison of our algorithm and k-means clustering. As described earlier, the k-means clustering does not perform well on the unbalanced datasets. The difference of accuracy between our algorithm and k-means clustering is huge in this case. The only dataset that k-means performs slightly better on than our algorithm is the S1 dataset. For A-Sets, the performance of all these algorithm does not vary much but our algorithm with euclidean distance metric has the better accuracy. Our algorithm with manhattan distance metric outperforms all in the Aggregation dataset, but it performs poorly for Flame dataset. In average, our algorithm with euclidean distance metric has the best accuracy among all.Fig. 7 shows the comparison plot of our algorithm with two different distance metrics and k-means clustering.

Then we run our algorithm with euclidean distance metric and k-means clustering on Birch2 dataset. We run the algorithm for $k = 10$ to 100 for an interval of 10 clusters. We plot the results and it is shown in fig. 8. We see that, the accuracy of k-means gradually decreases with increasing value of k which is also described in [9] that k-means works worse with increasing number of clusters. The performance of our algorithm also decreases with increasing k but still has a better accuracy than k-means.



(i) A point set (ii) Algorithm 7 output (iii) k-Means output

(iv) A point set (v) Algorithm 7 output (vi) k-Means output

(vii) A point set (viii) Algorithm 7 output (ix) k-Means output

(x) A point set (xi) Algorithm 7 output (xii) k-Means output

(xiii) A point set (xiv) Algorithm 7 output (xv) k-Means output

(xvi) A point set (xvii) Algorithm 7 output (xviii) k-Means output

Fig. 6: Some cases where k-Means clustering fails to generate proper clusters but our algorithm excels.

Table 1: Comparison of Our Algorithm and k-means clustering for the examples of Fig. 5.

Example	Total Points	Total mislabeled points		Accuracy	
		k-means	Alg. 7	k-means	Alg. 7
1	10	5	0	50%	100%
2	10	2	0	80%	100%
3	20	1	0	95%	100%
4	38	19	0	50%	100%
5	36	10	0	72.2%	100%
6	110	3	0	97.5%	100%

Table 2: Comparison of Our Algorithm and k-means clustering for different benchmark datasets.

Dataset	Total Points	k	Accuracy		
			Alg. 7		k-means
			euclidean	manhattan	
Aggregation	788	7	80.838%	85.787%	78.553%
S1	5000	15	87.02%	85.32%	82.04%
Unbalance	6500	8	94.169%	88.277%	68.108%
Flame	240	2	87.083%	73.333%	81.250%
A1	3000	20	85.067%	81.133%	83.2%
A2	5250	35	83.048%	78.038%	79.086%
A3	7500	50	84.013%	78.933%	81.907%

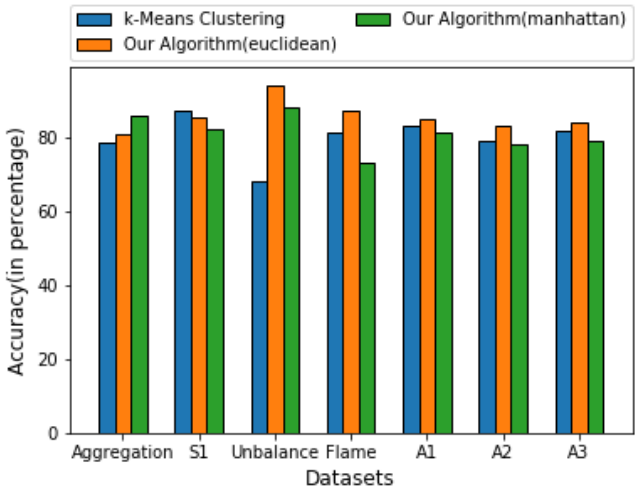


Fig. 7: Performance of our algorithm and k-means clustering for different benchmark datasets.

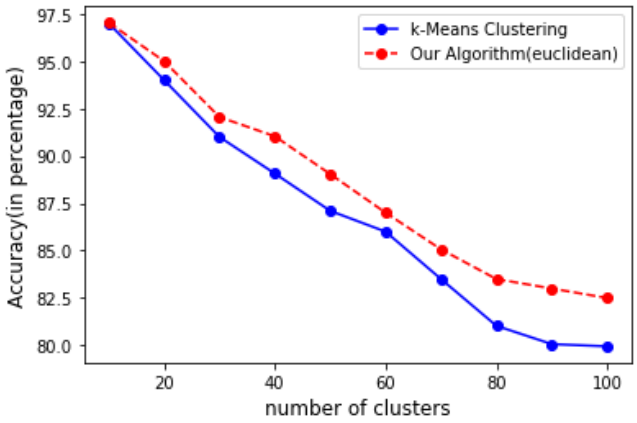


Fig. 8: Performance of our algorithm and k-means clustering for increasing number of clusters on Birch2 dataset.

5 Future Works

We focused on 2-dimensional points in this paper. But our algorithm can be extended for higher dimensions. Analyzing the running time and accuracy of our algorithm for higher dimensions would be an interesting problem.

We tested our algorithm for only 3 kinds of Minkowski distances(euclidean, manhattan and chebyshev). But the question remains is that will our algorithm work better for other similarity metrics. The cosine similarity is a widely used metric for data clustering. Mahalonobis distance is extremely popular in machine learning and pattern recognition these day. Euclidean distance does not differentiate between correlated and uncorrelated dimensions. But mahalonobis distance takes account of the correlation between the dimesions of the data and that is why it is widely used in machine learning these days.

Finding a better way to overcome the problem of decreasing accuracy with increasing number of clusters would be a great job. As the real world data is huge and can be really complex in structure, higher number of clusters are often needed to produce.

Finally, testing accuracy on different datasets can find more clues about how to increase the quality of our algorithm. Similarity based on distance measure really do not work on some datasets like some specific image data or audio data. In these case, we need to test our algorithm with completely new kind of similarity metrics and see whether these metrics actually work with the flow of our algorithm.

6 Conclusion

We presented a new clustering algorithm in this paper. Our algorithm is a divisive hierarchical algorithm. It starts by considering the whole input point set a single cluster and then recursively divides the cluster into smaller clusters. We focused on 2-dimensional points only in this paper. For maximum k clusters allowed and given an input point set consisting of n points, we proved that our algorithm has a running time complexity of $O(nk \log n)$.

Our algorithm has basically four steps those repeat in every iteration. It follows a greedy heuristic to complete one of these steps. After each iteration, it also follows that greedy heuristic to select a cluster from the remaining cluster to divide it and create two new clusters. It also has a selection criterion that selects the best clustering state among each value of k and finally outputs that state.

We compared our algorithm with the widely popular k-means clustering. We discussed the cases where k-means performs poorly. We also showed some cases experimentally where k-means clustering fails but our algorithm excels.

We compared our algorithm's performance with k-means clustering on some benchmark datasets. These datasets differ in size and shape. We tested the performance of both our algorithm(with 2 different distance metrics) and k-means clustering on these datasets and find that our algorithm with euclidean distance metric has the best accuracy among all these. We then tested both algorithms' accuracy with increasing value of k and find that our algorithm performs better than k-means clustering.

References

1. Arthur, D., Vassilvitskii, S.: How slow is the k-means method? In: Proceedings of the twenty-second annual symposium on Computational geometry. pp. 144–153 (2006)
2. Behsaz, B., Salavatipour, M.R.: On minimum sum of radii and diameters clustering. Algorithmica **73**(1), 143–165 (2015)
3. Berkhin, P.: A survey of clustering data mining techniques. In: Grouping multidimensional data, pp. 25–71. Springer (2006)
4. Bilò, V., Caragiannis, I., Kaklamanis, C., Kanellopoulos, P.: Geometric clustering to minimize the sum of cluster sizes. In: European Symposium on Algorithms. pp. 460–471. Springer (2005)
5. Charikar, M., Panigrahy, R.: Clustering to minimize the sum of cluster diameters. Journal of Computer and System Sciences **68**(2), 417–441 (2004)
6. Cormack, R.M.: A review of classification. Journal of the Royal Statistical Society: Series A (General) **134**(3), 321–353 (1971)
7. Delattre, M., Hansen, P.: Bicriterion cluster analysis. IEEE Transactions on Pattern Analysis and Machine Intelligence (4), 277–291 (1980)
8. Doddi, S.R., Marathe, M.V., Ravi, S.S., Taylor, D.S., Widmayer, P.: Approximation algorithms for clustering to minimize the sum of diameters. In: Scandinavian Workshop on Algorithm Theory. vol. 1851 of Lecture Notes in Computer Science, pp. 237–250. Springer (2000)

9. Fränti, P., Sieranoja, S.: K-means properties on six clustering benchmark datasets. Applied Intelligence **48**(12), 4743–4759 (2018)
10. Friedman, J.H., Bentley, J.L., Finkel, R.A.: An algorithm for finding best matches in logarithmic expected time. ACM Transactions on Mathematical Software (TOMS) **3**(3), 209–226 (1977)
11. Gibson, M., Kanade, G., Krohn, E., Pirwani, I.A., Varadarajan, K.: On metric clustering to minimize the sum of radii. Algorithmica **57**(3), 484–498 (2010)
12. Hansen, P., Jaumard, B.: Cluster analysis and mathematical programming. Mathematical programming **79**(1-3), 191–215 (1997)
13. Jain, A.K.: Data clustering: 50 years beyond k-means. Pattern recognition letters **31**(8), 651–666 (2010)
14. Lloyd, S.: Least squares quantization in pcm. IEEE transactions on information theory **28**(2), 129–137 (1982)
15. Monma, C., Suri, S.: Partitioning points and graphs to minimize the maximum or the sum of diameters. In: Graph Theory, Combinatorics and Applications (Proc. 6th Internat. Conf. Theory Appl. Graphs). vol. 2, pp. 899–912 (1989)
16. Toussaint, G.: Solving geometric problems with the rotating calipers. IEEE MELECON, 1983 pp. 1–8 (1983)
17. Vattani, A.: K-means requires exponentially many iterations even in the plane. Discrete & Computational Geometry **45**(4), 596–616 (2011)