Evaluation and Comparison of $k-$Means, $k-$Medoids, and Hierarchical Clustering via Self-Designed Data Sets

Xintong Zhu
Numerical Computing
Course Final Project
Instructor: Professor Margaret H. Wright

Abstract

Nowadays, living in a world filled up with data, numbers, and information, clustering analysis has become one of the most popular and essential field to explore. This paper mainly focuses on several most commonly applied and useful clustering methods. This paper first discusses background knowledge on several traditional clustering algorithm models, and then introduces some heuristic clustering algorithms that aim to enhance those conventional models. The paper then evaluates those clustering algorithm using self-designed data sets, which are generated to check and prove some specific features and drawbacks of each algorithm. Observations and conclusions are made; the most surprising result is that though performing quite bad with large and regular three-dimensional data set and generating poor and unbalanced dendrogram, the single linkage agglomerative hierarchical clustering is the one that is able to deal with unevenly distributed data sets, and data sets that generate specific shapes in geometry perfectly.

1 Introduction

Living in a world filled up with data, numbers, and information, clustering analysis has become one of the most popular and essential field to explore. Such kind of unsupervised algorithm enables machines to automatically classify a large amount of data, which provides people with a direct and clear visualization of the overall structure and distribution of data sets that probably contains millions of data.

Various clustering algorithms are developed for dealing with various situations, such as algorithms that are designed to deal with extremely large data sets, algorithms that are designed to deal with data sets containing obvious outliers, and even algorithms that can generate a tree-structure output for people to not only obtain clustering information but also get to know the data sets level by level, which facilitates their further exploration. Throughout the year, people are also devoting themselves to improve those pre-established traditional algorithms, intending to improve such kind of unsupervised algorithm and to let it behave more and more closed to the classifying works manually done by human beings.

With multiple clustering algorithms existed nowadays, there seems to be a necessity of a overall evaluation and comparison of those algorithms, which can further guide people which algorithm to use under in front of which kind of data set. This paper intends to realize such evaluation and comparison focusing on a few commonly used and practical clustering algorithm, including $k-$means++, PAM $k-$medoids, CLARA $k-$medoids, and four linkages within the agglomerative hierarchical clustering. The paper also includes a introduction of several other commonly used clustering algorithms, along with a detailed discussion and self-understanding.

This paper intends to evaluate algorithms by their advantages and disadvantages, and their clustering assignment features. In other words, this paper aims to analyze which kind of algorithm is suitable for which kind of data sets. Therefore, this paper uses data sets designed on my own to test and check features of each algorithm and evaluate performance of each algorithm on data sets with different features. Indeed, with the help of these self-designed data sets, some surprising (at leas to me) results are found, especially about the single linkage algorithm.

The paper would go as follows. Next section briefly introduces and discusses several related paper works. Background knowledge and discussion regarding with multiple clustering algorithms are included in Section 3-5. Evaluations are discussed in details in Section 6, including an introduction of all test data sets, experiment results, and analysis.

2 Related Works

This section is a summarization of knowledge learnt from previous related paper works. More details are discussed in later sections.

The first reading that I have done, which built up my foundation knowledge and attracted my attention to the clustering topic, is *Concise Machine Learning* by Jonathan Richard Shewchuk (2020). In Chapter 21 "The Singular Value Decomposition; Clustering", he

first introduces four utilities for clustering, which are discovery, hierarchy, quantization, and graph partitioning. He mainly explains three methods to cluster: the $k-$Means Clustering aka Lloyd's Algorithm by Stuart Lloyd in 1957, the $k-$Medoids Clustering, and the Hierarchical Clustering. Besides, The author includes a brief introduction on the two main kinds of algorithms for hierarchical clustering: the bottom-up agglomerative clustering and the top-down divisive clustering. Four clustering methods are explained for the agglomerative algorithm, which are complete linkage, single linkage, average linkage, and centroid linkage, and the author arrives at the conclusion that the average linkage and complete linkage are the most commonly applied methods.

An interesting paper work I found that intends to enhance existed clustering algorithm is the "Constrained $k-$means Clustering with Background Knowledge" by Kiri Wagstaff and Claire Cardie (2001). This paper mainly proposes an enhancement of the conventional clustering method. Given that in some cases, information of the problem domain is available, and therefore, it is feasible to develop some artificial constraints while running the clustering algorithm. Given that in many occasions, the number of clusters, i.e. k, is known, it is possible to generate instance-level constraints which include information about which two instances are supposed or not supposed to be in the same cluster. In other words, the team modified the $k-$means clustering from a pure unsupervised algorithm to a semi-supervised one. Six data sets with artificial constraints are tested, and the evaluation results indicate that this constrained $k-$means algorithm indeed improve the clustering performances. They also test their algorithm on unconstrained instances, and the enhanced clustering performance demonstrates that their algorithm is able to generalize the information of constraints to the unconstrained instances.

Multiple hierarchical clustering methods are evaluated in "Evaluation of Hierarchical Clustering Algorithms for Document Datasets" by Ying Zhao and George Karypis (2012). The paper evaluates various main partitional hierarchical clustering algorithms and agglomerative hierarchical clustering algorithms. Six clustering criterion functions for partitional algorithms cluster-bisection process' optimization are introduces, and three cluster selection schemes for agglomerative algorithm cluster-merging process' optimization are explicated. The paper test the different hierarchical clustering algorithm by six data sets, and compare the clustering results using relative FScore and dominance matrix. Both evaluation methods indicate that the partitional algorithm overall performs better than the agglomerative algorithm, and among the agglomerativ algorithms, the UPGMA method performs relatively the best. Moreover, the paper introduces the constrained agglomerative algorithm as well, and experiments results demonstrate that the constrained agglomerative algorithm, which use partitional algorithm to create constraints on conventional agglomerative algorithm and therefore combines advantages of both partitional and agglomerative algorithm, is even better than the partitional algorithm.

3 $k-$Means Clustering

Since under many situations, there is no ground truth for programmers to compare the clustering reults to an absolutely exact answer key, clustering is sometimes labeled as an unsupervised learning, which focuses on classifying data into group by the similarities between each pair in order to embody a general picture of the structure of data sets. Among various unsupervised clustering method, $k-$means is the most popular one not only because of its simplicity in algorithm structure, but also because of its efficiency and relatively fair accuracy.

In this section, background knowledge in $k-$means clustering is introduced and explained, along with some self-learning understanding and deduction.

3.1 $k-$Means Clustering aka Lloyd's Algorithm (Stuart Lloyd, 1957)

The very traditional and commonly used $k-$mean algorithms are based on Lloyd's Algorithm proposed in 1957. The goal is to partition data into k groups, i.e., cluster, given k as a known input, in which way data points in the same cluster are as similar as possible, while data points in different clusters are as different as possible [Dabbura, 2018].

The logic flow goes as follows. Given input points $\{x_1, x_2, ..., x_n\}$ and number of clusters to be assigned with $\{y_1, y_2, ..., y_k\}$, let n_i denote the number of data points in cluster y_i, and define the mean for cluster y_i to be

$$\mu_i = \frac{1}{n_i} \sum_{x_j \in y_i} x_j$$

and the algorithm attempts to minimize the square of errors, i.e.,

$$\sum_{j=1}^{k} \sum_{x_i \in y_j} ||x_i - \mu_i||^2 \tag{1}$$

To realize this goal, iterations are applied for following algorithm base model [Shewchuk, 2020].

Algorithm 1 Basic Model for $k-$Means Clustering

1: **repeat**
2: fix y_j, and update μ_j (1)
3: fix μ_j, and update y_j (2)
4: **until** step (2) does not change any data point assignment

In other words, the algorithm plans to alternate between 1) fix current cluster assignments, and recompute the cluster mean, and 2) fix cluster means,

and re-assign data points in order to minimize the square of error (Eq.1). This implies that the very basic alorithm model requires to try and check the results for every partitition, and therefore takes time complexity $O(nk^n)$.

To start with the algorithm, i.e., for step (1), there are mainly two feasible established methods to initialize μ_i [Shewchuk, 2020].

- Forgy method: pick k random data points as initial μ_i.
- Random partition: assign each data point to a cluster, with total k clusters, randomly, and then calculate the mean of the k randomly assigned clusters as the initial μ_i.
- k-means++: pick k random data points, but with each data point picked deliberately far from the other, i.e., a biased distribution of Forgy method

Practically, the k-means++ is the most commonly used one. My deducted reason for this is that Forgy and $k-$means++ have one step less than the random partition, and therefore are cheaper and more efficient; compared with Forgy, k-means++ is more reasonable and practical in the way that by deliberately picking centers far away from each other, it avoids the possibility that all means might be randomly chosen to be so closed to each other that they are all supposed to belong to the same cluster, which is possible in Forgy. Therefore, compared to Forgy, k-means++ is even more efficient. Note that the default Matlab function `kmeans` uses the k-means++ algorithm.

Based on above discussion, note that there are two points in which further progress and improvements can be realized. First, the optimization function in Eq.1 includes cluster means μ_i, which is changing in every iteration until when the current iteration does not change any assignment of data points. A more convincing function that does not require μ_i is proposed as follows

$$\sum_{i=1}^{k} \frac{1}{n_i} \sum_{x_j \in y_i} \sum_{x_m \in y_i} ||x_j - x_m||^2 \tag{2}$$

The advantage of above equation is that it does not include cluster mean, and therefore is more objective and convincing. It purely represents the average of sum of distances between all pairs of data points within the same cluster.

Another point here is that as we can see, the clustering results may depend heavily on initial choices of μ_i's. Therefore, for higher accuracy in reality, the algorithm is ran for several times, with each time picking different μ_i's.

3.2 Constrained $k-$Means Clustering aka COP k-means Clustering

The clustering method described in Section 3.1 is an unsupervised algorithm, i.e., data points are assigned and clusters are finally formed automatically, with no information about the problem domain available. Only the number of clusters k is required as an input (and methods to decide this parameter is discussed in next section). In many real cases, however, much more information about problem domain, in addition to data points (i.e., instances) themselves, is available initially. Therefore, instead of using greedy method to assign instances to the closest μ_i in every iteration, constraints can be made, which convert the algorithm to be semi-supervised but more flexible and efficient.

One obvious drawback of $k-$means++ is that since it uses greedy algorithm, it does not allow instances that are far away from each other to be assigned to the same cluster, even though they belong to the same cluster obviously if directly viewing the data sets scatter graph; similarly, it automatically assign instances that are relatively closed to each other to the same cluster, even though the two instances indeed have possibility to belong to different clusters originally. (Examples on this point is discussed more deeply in Section 6.) Such overcoming can be tackled down by setting instance-level constraints that express a priori information on which instances should or should not be assigned to the same cluster [Wagstaff, Cardie, 2001]. There are mainly two kinds of constraints.

- Must-link: two instances should be in the same cluster.
- Cannot-link: two instances should not be in the same cluster

Compared to the traditional $k-$means clustering algorithm, COP k-means clustering supplement another block of algorithm to check if the constraint. Instead of using greedy algorithm to always assign data points to their closest μ's, COP-clustering first sort the clusters by the distance between their centers and the to-be-assigned data point w_i in descending order. Denote as $C_1, C_2, ..., C_k$ (assume there are k clusters). Then, starting from the most closed one C_1, the algorithm checks if there is any data point $w_j, i \neq j$ in C_1 such that the must-link or the cannot-link is disobeyed for pair (w_i, w_j). If so, then continue on with the next cluster C_2; otherwise, assign w_i to C_1.

Still, let $C_1, ..., C_k$ be the initial k clusters, which can be chosen by methods described in Section 3.1. The algorithm is listed in Algo.2.

This algorithm, theoretically speaking, is indeed able to enhance accuracy of tradition $k-$means++. The drawback is that, however from my perspective, it is not always feasible. Artificial constraints, i.e. the must-link and cannot-link, can only be generated when certain information is available about the problem domain. A rough picture on the clusters information of a few instances is needed, which is not always feasible in real cases. Therefore, practically, COP $k-$means clustering is frequently realized by first apply a certain times

Algorithm 2 COP k-Means Clustering

function COP-k-MEANS(data set D, must-link $\Gamma_y \subset D \times D$, cannot-link $\Gamma_n = \subset D \times D$)
  repeat
    for each data point $d_i \in D$, sort the clusters by distance in descending order $C_1, ..., C_k$
    if VIO-CONSTRAINT($d_i, C_m, \Gamma_y, \Gamma_n$) **then**
      assign d_i to C_m
    else
      Continue on to check C_{m+1}
    end if
    Update means μ_i's for all C_i's by taking the average of all d_i's that have been assigned to it in current iteration
  until assignment does not change in further iteration (i.e., convergence)
  return $(C_1, ..., C_k)$
end function
function VIO-CONSTRAINTS(()data point d, cluster C, must-link Γ_y, cannot-link Γ_n)
  for $(d, d_y \in \Gamma_y$ **do**
    if $d_y \notin C$ **then return** true
    end if
  end for
  for $(d, d_n \in \Gamma_n$ **do**
    if $d_n \in C$ **then return** true
    end if
  end for
  return false
end function

of regular $k-$means algorithm, i.e. $k-$means clustering that does not involve constraints. Based on the results, certain instances are labelled by a parameter that contains certain information about their belonged clusters. This procedure can be realized by picking and labelling instances that does not change their belonged cluster dramatically during the previous runs of regular k-means clustering. Then, artificial constraints are able to be generated by randomly picking two instances from the data sets and checking their labels. If the two instances share the same label, a must-link is defined; otherwise, a cannot-link is constructed. If one (or both) instance is not labelled, repeat this procedure with a new randomly picked pair of instances from the data set.

Note that similar to $k-$means++, COP $k-$means in practical is ran for multiple trials, with different initial centers and probably different links for some instance pairs as well. Sometimes the result with the highest accuracy is chosen to represent, while sometimes the average of accuracies of all trials is taken to represent the overall accuracy of current test experiment performance.

Another important thing to note here is that in this section, we mainly use the default distance measure: Euclidean distance. In general, however, it is not always the case that means is the optimal distance metric. And in next section, we introduce a generalized version of the $k-$means clustering.

4 k-Medoids Clustering

In section, we discuss the background knowledge of a generalization of $k-$means clustering, and that is the $k-$medoids clustering.

From my perspective, there are two essential differences between $k-$medoids and $k-$means. First, instead of keeping Euclidean distance metrix, $k-$medoids encourages arbitrary distance measure, as long as the basic triangle inequality is satisfied. Two commonly used distance measures are l_1 norm and l_∞ norm, which are defined as

$$||x||_{l_1} = \sum_{i=1}^{n} |x_i|$$

$$||x||_{l_\infty} = \max_i |x_i$$

with $x = (x_1, x_2, ..., x_n)^T \in \mathbb{R}^n$

The reason for this is that in some cases, the Euclidean distance measure is not the optimal choice for clustering in order extract useful information from the data sets. One reasonable case in which l_∞ norm is more suitable than Euclidean distance is explicated in detail in Shewchuk's paper [2020].

Therefore, with other distance measures possibly applied, another clustering method, besides the $k-$means, is required, which is able to deal with distance measures such as l_1 and l_∞. The second difference is that instead of using clusers' means, here we use medoids, the instance that minimizes total distance to other instances in the current cluster. This implies that differing from $k-$means, in which case the cluster mean may be a data that does not belong to the original data set, here the definition of medoid guarantees that it must be one of the instances from the originial input data sets file. Therefore, this algorithm is object-based, which means that we use k instances (i.e., objects) from the input file to represent k clusters instead of calculating means to represent each cluster.

4.1 Partitioning around Medoids (PAM) algorithm

The most commonly applied algorithm for $k-$medoids is PAM algorithm, which stands for partitioning around medoid. Denote the cluster with m_i being the medoid to be M_i. And we use the word "cost" to represent the average dissimilarity within a cluster. The algorithm flow goes in Algo.3.

First note that clearly, PAM applies greedy algorithm, and the greedy search may not be able to guarantee optimal clustering assignment, which is the disadvantage of this algorithm. The advantage of this algorithm is that, on the other hand, the greedy search increases its speed, which makes it faster than exhaustive search. Besides, by only computing the change of

Algorithm 3 PAM $k-$medoids clustering

```
randomly select k instances as medoids
assign each instance to the closest medoid
repeat
    for each medoid m_i do
        for each instance d_j ∈ M_i do
            compute the change of cost if swap m_i and d_j
            if the change of cost is so far the best then
                remember current combination of m_i and d_j
            end if
        end for
    end for
    do the best swap of m_i and d_j
until the cost does not decrease for further iteration
```

cost in every combination, the running time is reduced further from re-computing the overall cost for every combination. Therefore, although it cannot guarantee optimal clustering, it is fast and simple to implement.

From my perspective, similar to $k-$means discussed in previous sections, the drawback of this algorithm can be compensated as well by running this algorithm on the input data sets several times with each time choosing different initial medoids. The greedy search problem, however, may not be easy to deal with. The same problem of $k-$means happen here as well, since the PAM would always assign instances to the current closest medoids, and would always preclude instances that are relatively far away from each other to be assigned to the same cluster, which may not always be the case in reality. This problem may be solved, in my opinion, by implementing constraints. Similar to the COP $k-$means, here we can also add instance-level constraints which express priori on which two instances must be in the same cluster, while which two instance must not be. We can still realize this goal by first running a certain times of regular PAM algorithm on the input file, and generate constraints by evaluating the running results. Then we can add these constraints to the regular algorithm by first checking if current assignment breaks the constraint before applying current assignment.

4.2 Clustering Large Applications (CLARA)

When dealing with relatively large data sets, the performance of PAM can be influenced by some extreme values. Therefore, we here introduce another commonly used $k-$medoids algorithm, especially for large data sets. The CLARA algorithm, standing for Clustering Large Applications, is designed to solve the potential scaling challenges posed by PAM by applying "sampling" strategy. The algorithm goes in Algo.4.

Still, for better performance, the algorithm is ran multiple times, each time picking different initial sample subsets.

Algorithm 4 CLARA k-Medoids Clustering

```
repeat
    randomly create multiple subsets with fixed size from original input file
    run PAM on each subset, and choose the corresponding k medoids
    assign instances of the entire data sets to clusters by choosing the closest medoid
until the medoids do not change for further iteration
```

5 Hierarchical Clustering

The $k-$means and $k-$medoids introduced in previous sections is indeed simple and efficient to implement. The drawback of greedy algorithm can be mitigated by running multiple trials with different initial conditions. There is another obvious problem, however, for both $k-$means and k-medoids. In many real life cases, the main difficulty for these two algorithms is to decide the value of k which represents the number of clusters and is required for mandatory as a parameter of the algorithm. Without a certain amount of information about data sets, k is sometimes not easy to tell. Some techniques to evaluate the decision of k is introduced in Section 6.

Such difficulty leads us to another clustering method, which differs much from both $k-$means and $k-$medoids in both algorithm design and output format, which is the hierarchical clustering. Matching its name, instead of aiming to simply assigning each instance to a cluster such that the average dissimilarities are minimized, hierarchical clustering focuses on providing a view of data at different levels (i.e., assigning instances to different "hierarchy") [Zhao, Karypis, 2002]. Along with clustering assignment, hierarchical clustering generates trees called *dendrograms* as well (with examples shown later). Such special form of output, from my understanding, enables users to visualize and deal with data sets better and more directly, especially for those relatively large input files. More importantly, it provides users with a sense of "exploration". Instead of directly displaying the clustering result as $k-$means and $k-$medoids, hierarchical clustering allows users to view the complete process of clustering. The dendrogram in effect can simulate a sorting procedure of disimilarities between pairs of instances, which facilitates users to deeply explore the data set.

In general, hierarchical clustering can be classified into agglomerative algorithms and partition algorithms. (Note that here we make clarification. In some papers, the classification is agglomerative algorithms and divisive algorithms. I have therefore done relatively many researches on this point, and arrive at the conclusion that the so-called "partition algorithms" and the so-called "divisive algorithms" are very similar in

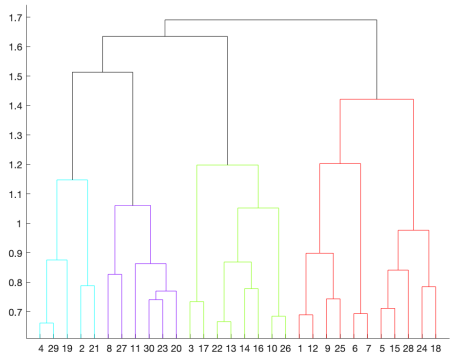

Figure 1: An example of dendrogram, and one way to cut and get clusters assignment, which is a test result in later Section 6

logic. This paper follows the terminology in Zhao and Karypis's paper in 2002, with the term "partitional".) And in the next few section, these two kinds of algorithms are introduced separately.

5.1 Dendrogram

Before entering to formal introduction of different hierarchical clustering algorithms, we first spend a short paragraph introducing dendrogram.

Dendrogram is a tree-structure representation, with leaves denoting all instances of the data sets. It is not the same as tree in data structure, though, in the way that, based on my understanding, it does not contain any internal node. The tree-shape intends to display the merging or spliting procedue (in each representing for agglomerative and partitional algorithms). In other words, dendrogram only needs to store 1) elements of initial clusters, and 2) the merging or splitting procedure in each iteration, and thus, there is no need to store elements of each cluster for every iteration (which would takes a lot of space). Hence, no internal node is needed. An example of dendrogram is provided in Fig.1

Note that in Fig.1, only the vertical axis is labelled. This is because only the vertical axis, or to say height, is meaningful in this context. The vertical height in dendrogram represents the dissimilarity (i.e., the distance) between two clusters. The horizontal axis, howevr, does not have essential meaning here. Swapping two subtrees from left to right of the same root is legal, and makes no changes to both the meaning of dendrogram and clustering result.

One possible drawback of hierarchical clustering is that traditionally, the algorithm generates and returns a dendrogram instead of an explicit clustering assignment as $k-$means and $k-$medoids. The way to obtain cluster assignment is to "cut" the dendrogram by a horizontal line. This procedure is frequently based on some criterion, or direct analysis and viewing from the dendrogram shape. Therefore, compared to $k-$means and $k-$medoids, hierarchical clustering is more subjective and flexible. In R language, we can use comman `cutree()` and `hclust()`. In Matlab, we can use `dengrogram()` to generates a dendrogram plot of the hierarchical binary cluster tree, and `cluster()` to extract clusters information from a hierarchical cluster tree.

Here in Fig.1, observe that the dendrogram is cut into four clusters, with each cluster drawing in a different color.

Now we are ready to discuss algorithms.

5.2 Agglomerative Hierarchical Clustering

Just as its name, agglomerative clustering generates the solution from bottom to up. Ir first splits each instance to its own cluster (i.e., at the initial step, each cluster contains only 1 instance). Then it starts iterations and merges two clusters once in every iteration by certain cluster selection schemes, which is introduced soon in this section. The iteration stops when all instances are in the same cluster (i.e., at the final step, there is only 1 cluster that contains all instances). A dendrogrom is built from bottom to up in this process.

There are mainly 5 commonly used cluster selection schemes. Here let $d(x, y)$ denote the distance metric. And $C_1, C_2, ..., C_m$ to be m clusters.

- *Single Linkage*: use the maximal similarity between instances from different cluster to represent the cluster similarity, i.e.,

$$sim_{single}(x_i, x_j) = \max_{x_i \in C_i, x_j \in C_j, i \neq j} d(x_i, x_j)$$

- *Complete Linkage*: use the minimal similarity between instances from different cluster to represent the cluster similarity, i.e.,

$$sim_{complete}(x_i, x_j) = \min_{x_i \in C_i, x_j \in C_j, i \neq j} d(x_i, x_j)$$

- *Average Linkage aka UPGMA*: use the average of pairwise similarities of the instances from each cluster to represent the cluster similarity, i.e.,

$$sim_{ave}(x_i, x_j) = \frac{1}{|C_i||C_j|} \sum_{x_i \in C_i, x_j \in C_j, i \neq j} d(x_i, x_j)$$

- *Centroid Linkage*: use the distance between two means from each cluster to represent the cluster similarity, i.e.,

$$sim_{centroid}(x_i, x_j) = d(\mu_i, \mu_j)$$

- *Ward's Method*: intends to minimize the total within-cluster variance, i.e., at each iteration, the method finds two clusters that generate minimal increase in total within-cluster variance after being merged. This method is recursive, with initial distance (i.e., similarity) between clusters to be defined as

$$sim_{ward}(C_i, C_j) = ||c_i - c_j||^2$$

with c_i and c_j represents for the initial single instance in cluster C_i and C_j.

In other words, the five cluster selection schemes are used to define similarity between clusters.

Here note that the centroid linkage can be applied reasonably only when Euclidean distance is used. Here the case is similar to the discussion of $k-$means and $k-$medoids. Again, when other distance metric is used, instead of using centroid, we can use medoid.

5.3 Partitional Hierarchical Clustering

Instead of building dendrogram from bottom to the top, partitional hierarchical clustering generates the tree from top to bottom. It works from an opposite direction to the agglomerative algorithms (just as their names).

Partitional clustering algorithm first partitions all instances into two clusters, which is done by optimizing a certain clustering criterion function. Then, from the two bisected clusters, the one with more instances if picked and bisected into two clusters again by optimizing a certain clustering criterion function. These procedures are repeated until when all instances are in different clusters, i.e., the data sets are splitting into clusters that all contain a single instance. In this way, the dendrogram is built, but this time from top to down, with leaves storing one single instance.

Basically, there are 3 commonly applied criterion function. Again, let $C_1, C_2, ..., C_m$ denote the m clusters, and $d(x, y)$ denote the distance metric.

- *Internal Criterion Function*:

$$F_{in} = \sum_{s=1}^{m} |C_s| (\frac{1}{|C_s|^2} \sum_{x_i, x_j \in C_s} d(x_i, x_j)) \quad (3)$$

  This criterion intends to maximize the sum of the average pairwise similarities between instances in each cluster. The criterion can also be written as

$$F'_{in} = \sum_{s=1}^{m} \sum_{x_i \in C_s} d(x_i, \mu_s) \quad (4)$$

  i.e., to maximize the sum of the average pairwise similarities between instances and its center of each cluster. Since this function involves cluster means, euclidean distance needs to be applied in this function.

- *External Criterion Function*:

$$F_{ex} = \sum_{s=1}^{m} |C_s| d(C_s, C) \quad (5)$$

  , with C denoting the centroid of the entire data set.

- *Graph Based Criterion Function*:

$$\sum_{s=1}^{m} \frac{cut(C_s, C - C_s)}{\sum_{x_i, x_j \in C_i} d(x_i, x_j)} \quad (6)$$

After introducing the criterion function, we go back to explain the initialization procedure. Given a data set, we initialize the clustering process as follows. First randomly pick two instances. Let the two instances represent two clusters. Then assign each instance to the closest cluster based on one of the criterion functions discussed above. The algorithm then continues by repeating such process on each of the newly generated cluster. During every iteration, for each instance d_i, the change of value of criterion function is computed if move d_i from current cluster to another cluster C_j. If any move (d_i, C_j) increases the value of the function, the algorithm moves d_i from its current cluster to cluster C_j. Otherwise, keep d_i in current cluster. Such process if repeated until each instance is in its cluster alone which contains a single instance.

6 Evaluation

We are now ready for experiments and evaluations. This paper mainly evaluates and compares following clustering algorithms:

- k-means++ clustering
- PAM Clustering
- CLARA Clustering
- Agglomerative Hierarchical Clustering
  - Single Linkage
  - Complete Linkage
  - Average Linkage
  - Ward's Method

For clarification, this paper is not going to evaluate all methods introduced above, because of limited time and pages.

Our evaluation process tests 6 data sets designed on my own, aiming to test and prove several features and understandings of certain algorithms. Name and descriptions of 6 data sets are in Table.1. For later convenience, an index is tagged for each data set, and later in this section, we use the index to refer to each data set.

I encourage you to view this section on computer instead of printed paper, since many plots involve colors for a better visualization.

6.1 k-Means++ Evaluation

For $k-$means clustering, this paper picks $k-$means++ to evaluate and compared with the other, simply because this is the most commonly used and practical models among others of $k-$means.

We use the command `kmeans` in Matlab for our evaluation. As discussed in Section 3.1, theoretically speaking, the clustering results of $k-$means++ depends on initial choice of cluster means, and in reality, multiple trials are conducted and the optimal (or average) clustering result is picked as the final result. Here first use R^3_{10000} data set to observe and check this statement.

Name & Index	Description
R^3_{10000}	10000 randomly generated data points in three dimension
R^2_{1000}	1000 randomly generated data points in two dimension, with purposeful uneven density distribution
R^2_{40000}	40000 randomly generated data points in two dimension
L^2_{30}	30 data points that form two parallel and same-length lines in two dimension
S^2_{200}	200 data points in two dimension that forms a specific special shape (such as two disjoint half circles)
S^2_{500}	500 data points in two dimension that forms a relatively dense specific special shape

Table 1: Names and descriptions of data sets used for experiments and evaluations

Trial	C_1	C_2	C_3	C_4	Average Distance
1	$(0.4170, 0.7203, 0.0001)^T$	$(0.3023, 0.1468, 0.0923)^T$	$(0.1863, 0.3456, 0.3968)^T$	$(0.53880.41920.6852)^T$	296.2655
2	$(0.4360, 0.0259, 0.5497)^T$	$(0.4353, 0.4204, 0.3303)^T$	$(0.2046, 0.6193, 0.2997)^T$	$(0.2668, 0.6211, 0.5291)^T$	297.1809
3	$(0.5508, 0.7081, 0.2909)^t$	$(0.5108, 0.8929, 0.8963)^T$	$(0.1256, 0.2072, 0.0515)^T$	$(0.4408, 0.0299, 0.4568)^T$	297.1602
4	$(0.9670, 0.5472, 0.9727)^T$	$(0.7148, 0.6977, 0.2161)^T$	$(0.9763, 0.0062, 0.2530)^T$	$(0.4348, 0.7794, 0.1977)^T$	**295.8884**
5	$(0.2220, 0.8707, 0.2067)^T$	$(0.9186, 0.4884, 0.6117)^T$	$(0.7659, 0.5184, 0.2968)^T$	$(0.1877, 0.0807, 0.7384)^T$	299.2823
6	$(0.8929, 0.3320, 0.8212)^T$	$(0.0417, 0.1077, 0.5951)^T$	$(0.5298, 0.4188, 0.3354)^T$	$(0.6225, 0.4381, 0.7359)^T$	298.2927
7	$(0.0763, 0.7799, 0.4384)^T$	$(0.7235, 0.9780, 0.5385)^T$	$(0.5011, 0.0721, 0.2684)^T$	$(0.4999, 0.6792, 0.8037)^T$	**295.8884**
8	$(0.8734, 0.9685, 0.8692)^T$	$(0.5309, 0.2327, 0.0114)^T$	$(0.4305, 0.4024, 0.5227)^T$	$(0.4784, 0.5554, 0.5434)^T$	295.8884
9	$(0.0104, 0.5019, 0.4958)^T$	$(0.1338, 0.1421, 0.2186)^T$	$(0.4185, 0.2481, 0.0841)^T$	$(0.3455, 0.1668, 0.8786)^T$	299.2131
10	$(0.7713, 0.0208, 0.6336)^T$	$(0.7488, 0.4985, 0.2248)^T$	$(0.1981, 0.7605, 0.1691)^T$	$(0.0883, 0.6854, 0.9534)^T$	297.1602

Table 2: Initial cluster means and average distance for 10 trials

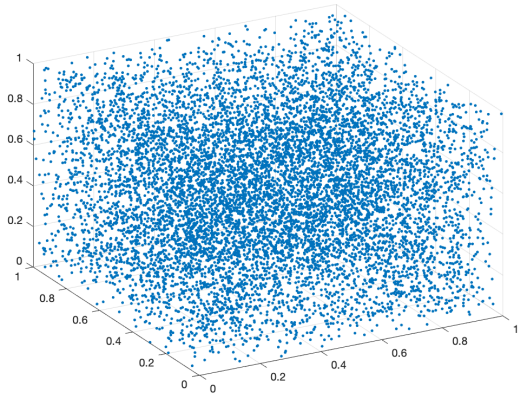

Figure 2: Raw data sets plot for R^3_{10000}

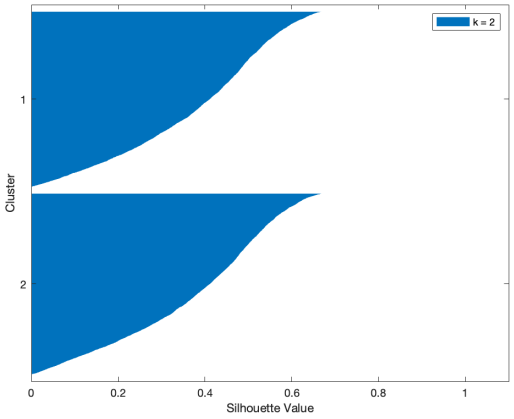

Figure 3: Silhouette when $k = 2$

We first decide the relative optimal choice of k, the number of clusters to be assigned. And we use the **silhouette** analysis to make such decision. The raw data set is plotted in Fig.2, and we evaluate cases when $k = 2$, $k = 3$, and $k = 4$, with corresponding silhouette graph plotted in Fig.3, 4, 5.

From the silhouette plots, we can see that $k = 2$ leads to the worst clustering performance, with the silhouette value averagely the smallest, approximately 0.63 for each cluster. $k = 4$ leads to the relatively best performance, with one cluster's silhouette value about 0.8, and the rest of the rest have silhouette value a little bit less than 0.8. Therefore, our experiment continues with $k = 4$.

We run the algorithm with parameter $k = 4$ and data set R^3_{10000} on 10 trials with different initial cluster means chosen for each trial. For objectivity, the 10 groups of different cluster means are generated ran-

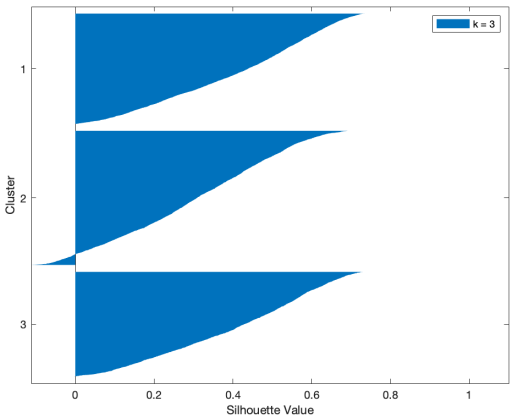

Figure 4: Silhouette when $k = 3$

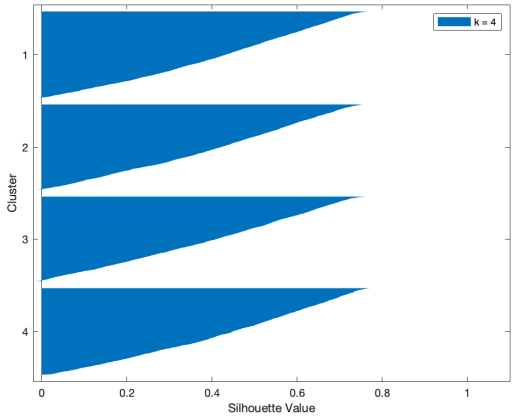

Figure 5: Silhouette when $k = 4$

domly. We evaluate the performance for each trial by calculating the average distance between each point to its corresponding cluster mean. We list the initial choice for cluster means and average distance results for each trial in Table.3, and display the clustering result in Fig.7-16.

From Table.2, we can see that indeed, with different initial cluster means, the clustering results vary. The best performance occurs at trial 4 and 7, with the least average distance about 295.8884. However, note that the four initial cluster means are not the same in these two trials. Therefore, we can see from this observation that although the clustering results depend on initial cluster means, there is no deterministic relation between the initial cluster means and clustering result performance. There can be cases when two different groups of initial cluster means finally lead to similar clustering result. One point to note here is that viewing from the clustering plots of Trial 4 and 7, which are Fig.10 and Fig.13, we can see that the cluster assignment of these two trials are very similar to each other (although plotted by different color assignments), which corresponds to the result that these two trials have similar average distance of between each point to its corresponding cluster mean.

Generally speaking, I will say the performance for $k-$means++ on this data set is relatively fair. Through Fig.7-16, we can see that $k-$means++ largely "split" the data sets in to 4 blocks, which are tagged by different colors. Since R^3_{10000} is generated within range $(0, 1)$, we can view the coordinates in this case as a cube with side length 1, which can be viewed imagined directly from Fig.7-16, and the $k-$means++ roughly generate clustering by assigning data points closed to one side of the cube or one side edge of the cube to the same cluster. For example, in Trial 4 Fig.10, we can see that the purple block is clustered around the outer top edge; the green block is clustered around the outer bottom edge; the yellow block is clustered around the inner left edge; and the blue block is clustered around the inner right edge. In Trial 10 Fig.16, the purple block is clustered closed to the outer right side; the green block is clustered closed to the outer right side; the yellow block is clustered closed to the inner left side; and the blue block is clustered closed to the inner right side. Though clustering in slightly different way, $k-$means++ overall assigns data points reasonably, automatically generating clustering results that are consistent to the structure of the data set.

The optimal result (with the optimal choice for cluster means) is shown in Fig.16, with the minimized average distance 0.4983.

The above experiment and analysis is just for a direct understanding check. Since now we are ready to evaluate some specific features of $k-$means++, we use a two-dimensional data set, which can be viewed more clearly. We now evaluate $k-$means++ with another data set, R^2_{1000}, which is generated randomly but purposefully unevenly. The raw data set is plotted in Fig.17.

The clustering result is plotted in Fig.17. Note that in this experiment, we choose initial cluster means by `kmeans` command in Matlab defautly. Moreoever, we also point out the mean for each cluster this time, which is labelly by a relatively bigger circle in Fig.18.

In this case, we pick $k = 3$. This is a reasonable choice since I purposefully generate the data set R^2_{1000} unevenly in density in three relatively disjoint blocks, with Matlab commands

```
X1 = 0 + rand(50,2) * 10;
X2 = 30 + rand(250,2) * 30;
X3 = 100 + rand(700,2) * 100;
```

, with `X1, X2, X3` denotes the three blocks. On purpose, we generate `X1` with 50 instances, `X2` with 250 instances, and `X3` with 700 instances. And the uneven distribution in density can be directly observed from Fig.16.

Therefore, the number of cluster is chosen to be 3, and naturally, we expect to see a cluster assignment consistent with the density distribution, which makes most sense in our common sense. However, the result turns out to be that $k-$means++ does not assign instances to clusters as what we think it supposed to be. In Fig.17, we can see that the algorithm assigns `X1` and `X2` to the same cluster, but splits `X3` to two clusters. And observe that the red cluster and blue cluster both contain instances quite closed to the other cluster, while the green cluster contains instances that are relatively far from the other.

This "weird" assignment can be explained if we consider the working principle of $k-$means++. As discussed in Section 3.1, $k-$means++ intends to minimize the within-cluster variation, and therefore the algorithm would focus more on bigger clusters than smaller clusters, i.e., the algorithm puts more weights on cluster that contains more instances. This leads to the result that some instances in small clusters may be left away from the cluster mean in order to guarantee that

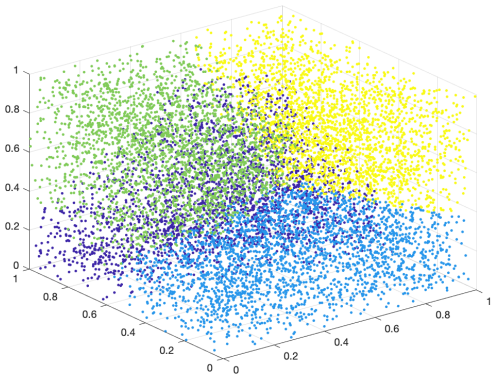

Figure 7: Trial 1 for R^3_{10000}

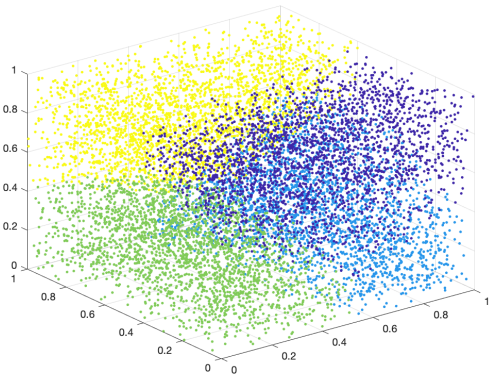

Figure 8: Trial 2 for R^3_{10000}

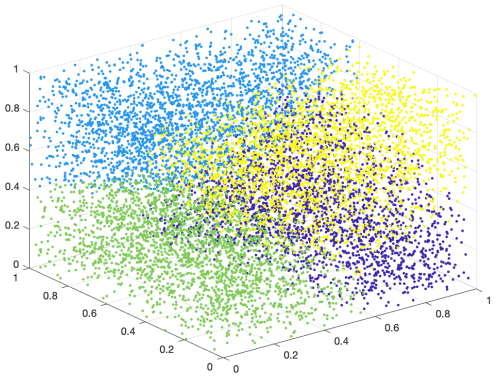

Figure 9: Trial 3 for R^3_{10000}

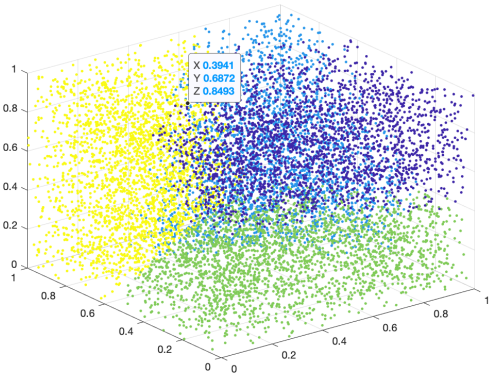

Figure 10: Trial 4 for R^3_{10000}

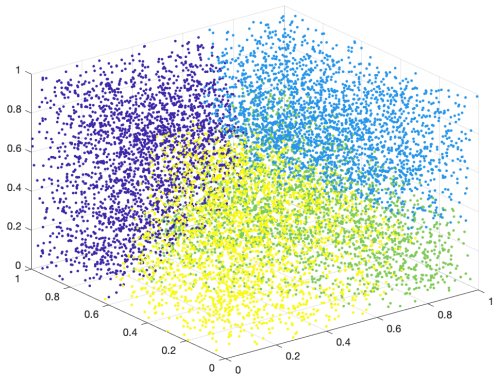

Figure 11: Trial 5 for R^3_{10000}

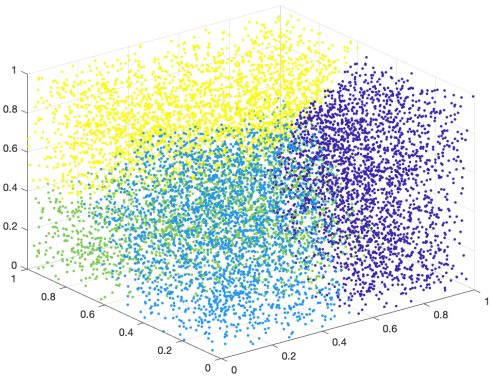

Figure 12: Trial 6 for R^3_{10000}

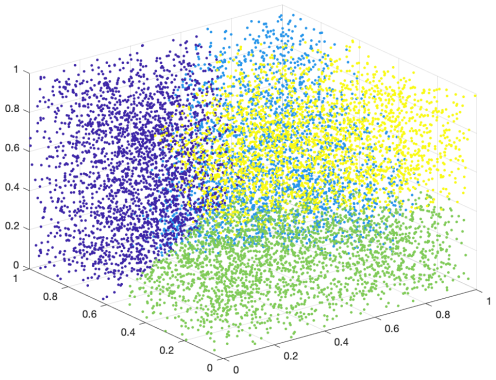

Figure 13: Trial 7 for R^3_{10000}

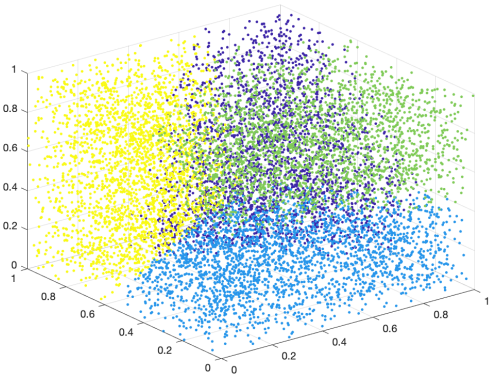

Figure 14: Trial 8 for R^3_{10000}

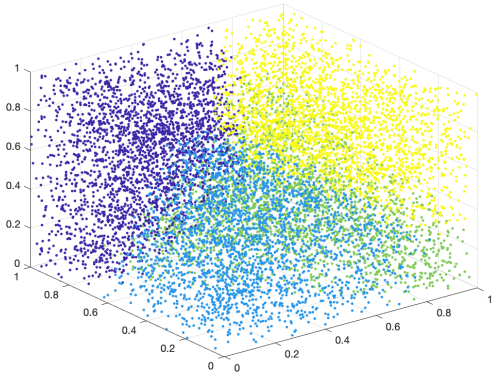

Figure 15: Trial 9 for R^3_{10000}

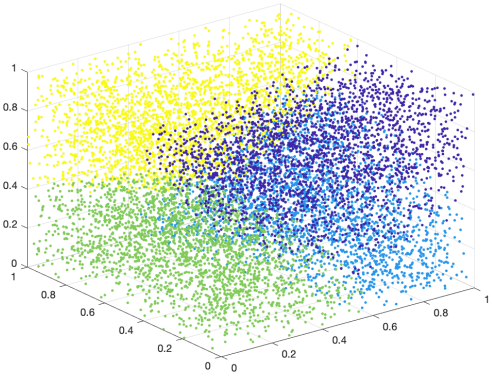

Figure 16: Trial 10 for R^3_{10000}

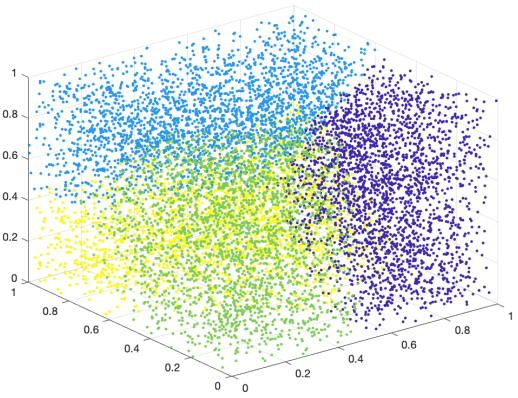

Figure 16: Optimal clustering result for R^3_{10000} for $k-$means++

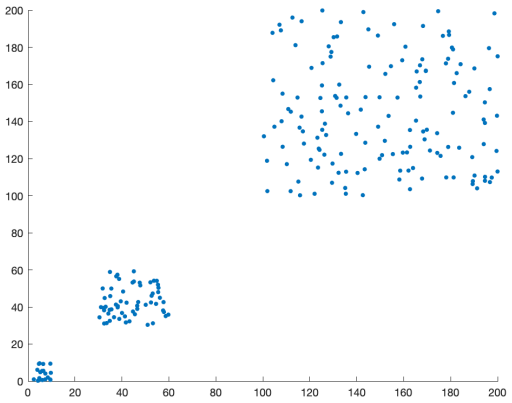

Figure 17: Raw data for R^2_{1000}

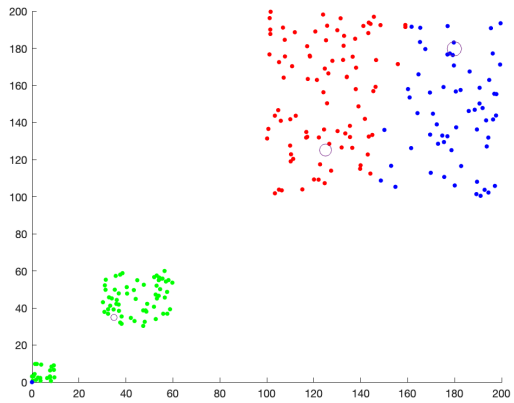

Figure 18: $k-$means++ clustering result for R^2_{1000}, along with the mean for each cluster

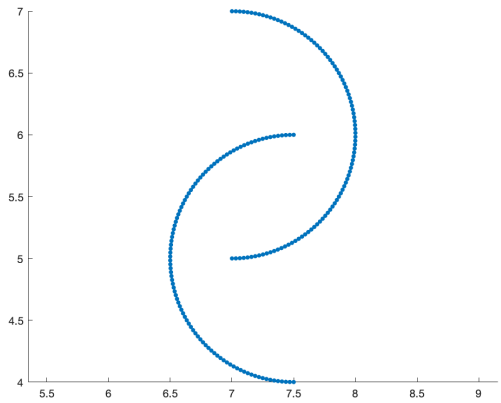

Figure 20: Raw data set for S^2_{200}

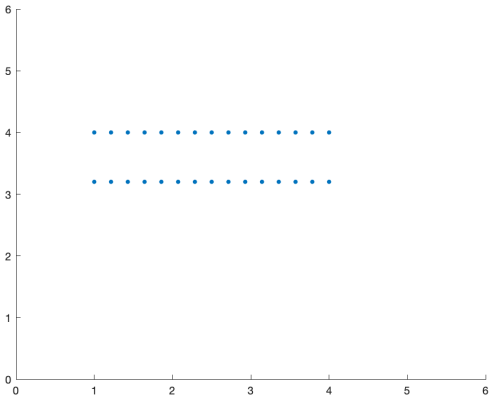

Figure 19: Raw data set for L^2_{30}

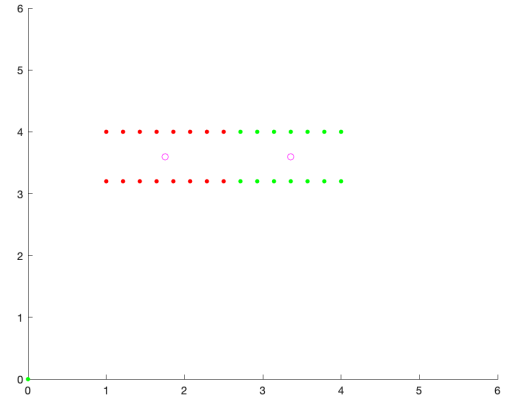

Figure 21: $k-$means++ clustering result for L^2_{30}

the within-cluster variance in bigger cluster is small enough. In other words, $k-$means++ scarifies smaller cluster in order to obtain "better" performance in bigger cluster.

We now test $k-$means++ with another two designed data sets L^2_{30} and S^2_{200}. These two data sets are all generated in two dimension, and intend to form some specific geometric shape. L^2_{30} is generated to form two parallel lines, and S^2_{200} is generated to form two disjoint half circles. The raw data plot is in Fig.19-20. Note that the intention for these two data sets is to test the performance of $k-$means++ when dealing with some specific geometric shape whose cluster assignment is easy and obvious to tell in advance. i.e., for L^2_{30} in Fig.19, observe that a cluster assignment that is consistent to common sense is to assign instances to 2 clusters, each cluster composing 1 line; for S^2_{200} in Fig.19, a reasonable cluster assignment is, still, assigning instances to 2 clusters, each cluster composing a half circle.

The clustering result is shown in Fig.21-22.

Observe that for L^2_{30}, $k-$means++ does not cluster the data set as we expect it do. Instead of splitting the data sets into two parallel lines, the algorithm cluster the data set into left and right blocks. It, indeed, makes sense to some extents, however. One weird point may be that at the splitting edge of the red and green blocks, note that the two adjacent points on the left and right hand side of the splitting edge is very closed to each other. However, the algorithm splits them into 2 clusters, through the distance between that two points is much smaller than the distance between two parallel lines. Such assignment can, still, be explained by the working principle of $k-$means++. First note that for each horizontal line, the distance between the leftmost point and the rightmost point indeed is bigger than the distance between the two parallel lines. Therefore, it makes sense to some extents that the algorithm does not assign the leftmost point and rightmost point of the same line to the same cluster. Moreover, keep in mind that the intention of $k-$means++ is to minimize the square error. Therefore, it chooses such assignment that the with-in cluster variances of two clusters are the same, in which way minimizes the overall square error. This can explain why the vertical splitting line is drawn

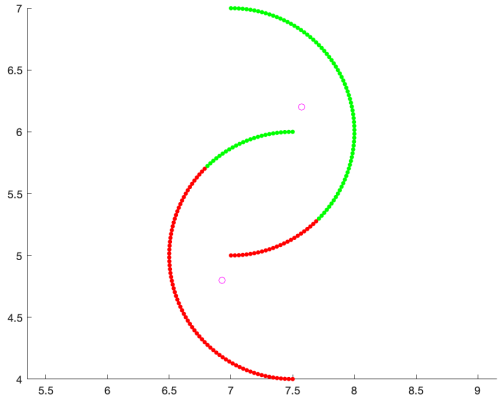

Figure 22: $k-$means++ clustering result for S^2_{200}

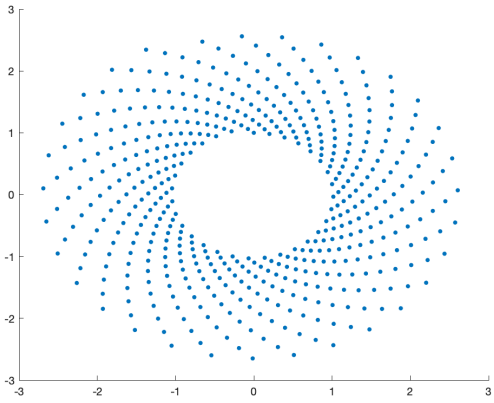

Figure 23: $k-$means++ clustering result for S^2_{500}

in the middle point.

On the other hand, however, the clustering result for S^2_{200} is even more surprising. Instead of assigning each half circle as a cluster, the algorithm splits each half circle. This result can be explained by similar reasoning as above.

One interesting point to note here is that, $k-$means++ does not fail in all regular geometric shapes' clustering. There indeed exists shapes that can be clustered by $k-$means++ as what we imagine it supposed to do. Consider the denser plot in Fig.23, and the corresponding clustering result in Fig.24.

S^2_{500} data set forms a certain number of netted ellipse, with points density decreased from the inner ellipse to the outer ellipse. It is a symmetric geometric shape. From Fig.23, we can see that the $k-$means++ "successfully" clusters the data sets in the way that is consistent to our common understanding (or to say, imagination). For $k = 2$, it cuts the the graph evenly. (Note that with different choice of initial cluster mean, the splitting line "rotates", fixing its middle point at the center of the ellipses, i.e., it always cuts the shape evenly and intersects the ellipses' center, but may at

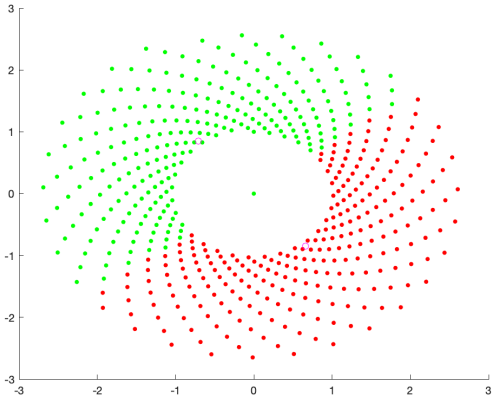

Figure 24: $k-$means++ clustering result for S^2_{500}

different angles.) The reason for this may be that comparing with L^2_{30} and S^{400}, this data set forms a shape that is much more regular and symmetric in the sense of geometry. Therefore, when minimizing the within-cluster variance, the algorithm is able to splits the shape evenly and guarantees the relative completeness in shape of each cluster simultaneously.

6.2 PAM & CLARA

By the evaluation in last section, we can see that $k-$means++ is indeed not perfect, and actually contains several problems. The experiment results are largely consistent with the theory discussion in Section 3. Bearing the drawbacks of $k-$means++ described above, we now evaluate the $k-$medoids, which is kind of similar to the $k-$means. Here, as discussed in Section 4.1 and 4.2, we mainly evaluate the PAM (Partitioning around Medoids) and CLARA (Clustering Large Applications).

To get an initial feeling, we, still, first uses R^3_{10000} to test the two algorithms. We use the Matlab command `kmedoids`, which has an attribute `Algorithm` that contains the implementation of `pam` and `clara`. The clustering results for PAM and CLARA are in Fig.25-26. We, similarly, use the average distance between points to their medoids (not centroids this time) to determine their performances. The average distances are written in the caption of each figure.

Through the average distance, we can see that in this case, CLARA performs better than PAM, which is reasonable and consistent to the theory discussion in Section 4, since CLARA is designed to deal with relatively large data sets. However, not that both of these algorithm perform worse than the $k-$means++ in this case (since the average distance of $k-$means++'s result is less than both of these two algorithms).

We then evaluate the two algorithms by data sets applied in previous section for $k-$means++, just in order to see how these two $k-$medoids algorithms work for cases that $k-$means++ does not perform very well. The results for the two algorithms on data sets R^2_{1000}

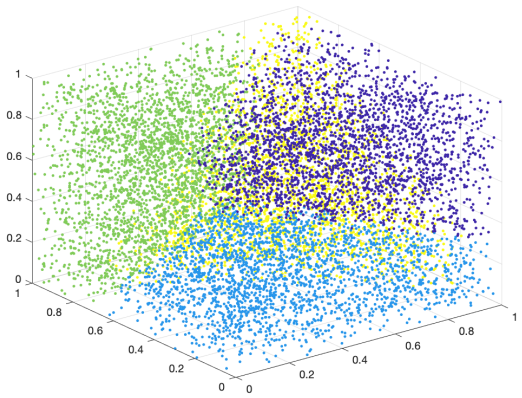

Figure 25: PAM clustering result for R^3_{10000}, with average distance 0.5003

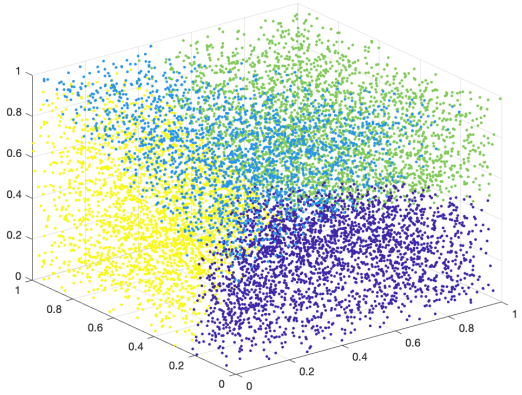

Figure 26: CLARA clustering result for R^3_{10000}, with average distance 0.4745

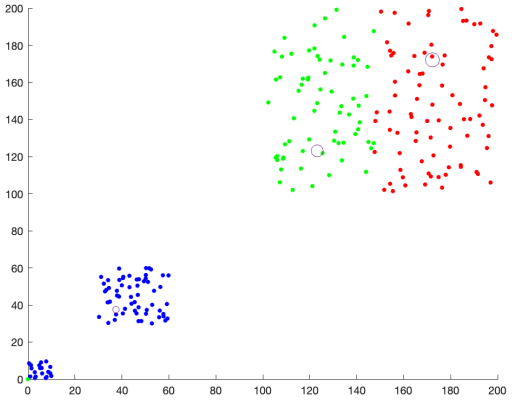

Figure 27: PAM clustering result for R^2_{1000}

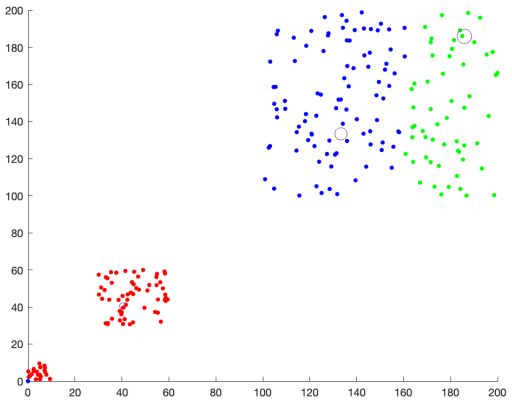

Figure 28: CLARA clustering result fo R^2_{1000}

are shown in Fig.27-28.

The result is kind of surprising, at least to me, since based on my understanding towards the $k-$medoids, it should be more reliable than $k-$means++ when dealing with noise and outliers since it intends to minimize the sum of general pairwise distances instead of within-cluster variances. The experiment result turns out, however, to be that both $k-$medoids algorithm do not deal with this data set well either. I will explain this result as follows. Similar to $k-$means++, $k-$medoids' aim is to minimize some attribute in an overall picture. Therefore, even though the attribute it intends to minimize is different from that of $k-$menas++, both of them weights more on bigger clusters than on smaller one. In other words, $k-$medoids still chooses to scarify the smaller cluster in order to guarantee that the pairwise distances in bigger cluster is small enough. It is indeed theoretically true that $k-$metoids is more robust when dealing with outliers. In this case, however, there is actually no outlier in the sense that all instances, though in different ranges, are generated in group. There is no instance that is generated in isolation. Therefore, in general, $k-$medoids do not preform obviously better than $k-$means++ in this case, i.e., both kinds of algorithms cannot deal with uneven density data sets very well.

We then evaluate L^2_{30} and S^2_{400}. The results are for the two algorithms are shown in Fig.29-32. Observe that in general, the results for the two algorithm are quite similar. This is consistent to the fact that the work principle for CLARA is just to run PAM on multiple subsets of original data sets iteratively. Therefore, their results would differ when the data set is big or there are outliers. For small data sets and data sets that do not contain obvious outliers, however, the two algorithms work very closed to each other, just as the results in these two cases. However, one obvious difference between the results for these two algorithms and $k-$means++ is that note that for both of the two data sets L^2_{30} and S^2_{400}, the cluster means (i.e., centroids) of $k-$means++ is some points that do not belong to the original data sets. For both of these two algorithms, however, the medoids are some points originally in the data sets. This corresponds to the prin-

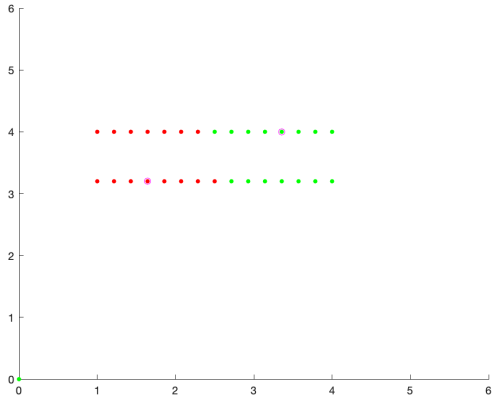

Figure 29: PAM clustering result for L_{30}^2

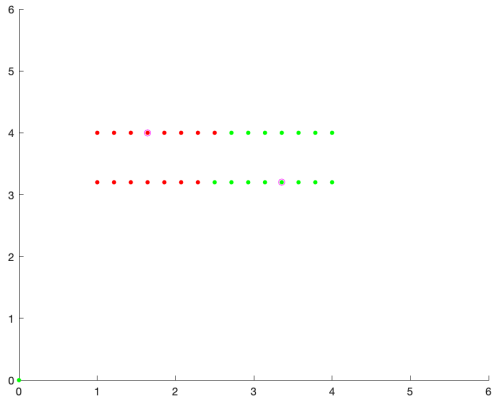

Figure 30: CLARA clustering result for L_{30}^2

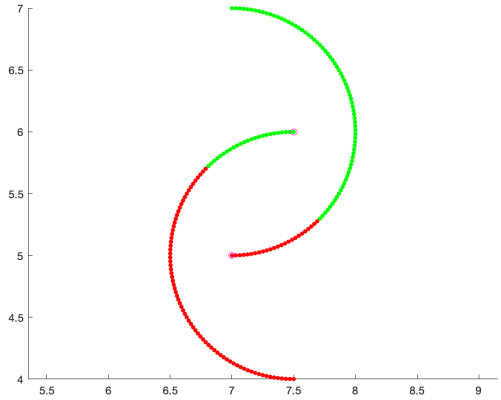

Figure 31: PAM clustering result for S_{400}^2

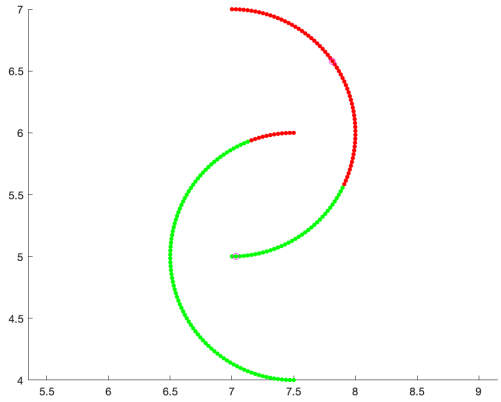

Figure 32: CLARA clustering result for S_{400}^2

ciple and definition of medoids, and also reminds us for the fact that $k-$medoids is an object-based clustering algorithm. Another interesting difference to note here is that for S_{30}^2, PAM evaluates it quite similar to $k-$means++, while CLARA's evaluation results is different from both of them. The most obvious difference here is that CLARA does not split the data set evenly as PAM and $k-$means++ do. My deduction for this phenomenon is that since CLARA runs PAM multiple times on subsets of original data set, some problems may happen when dealing with figures that contains some "intersection" (i.e., figures that are not in regular shape). The initial split of subset may influence the clustering result a little bit more dramatically in this case.

6.3 Agglomerative Hierarchical Clustering

After relative comprehensive experiments and analysis of $k-$means and $k-$medoids, we are now ready to start with evaluation of agglomerative hierarchical clustering. Different from the above three algorithms' evalutions, in this section, instead of focusing on some numerical values (such as average distance), we put our attention on the dendrogram analysis, along with the cluster assginment for sure as well.

In this section, we mainly evaluate complete linkage, single linkage, average linkage, and Ward's method. We use the Matlab command `linkage`, `dendrogram`, and `cluster` to do the experiment codes. Matlab implements the above linkages by an attribute inside the function `linkage`, with specific command `single`, `complete`, `average`, and `ward`.

First note that there is another linkage that is not mentioned in previous Section 5.2, which is the median linkage. We did not include the median in the discussion of cluster selection schemes for agglomerative clustering, and this is because this method is approriate to Euclidean distances only. It measures the weighted center of mass distance.

In this section, we first evaluate and compare these four linkages. We again use R_{10000}^3 to test the four linkages, and generate the clustering and in Fig.34-41.

Several comments are needed here.

First, observe that the single linkage in this case behaves extremely weird. Instead of coloring the three-dimensional data set into 4 blocks, it seems that under

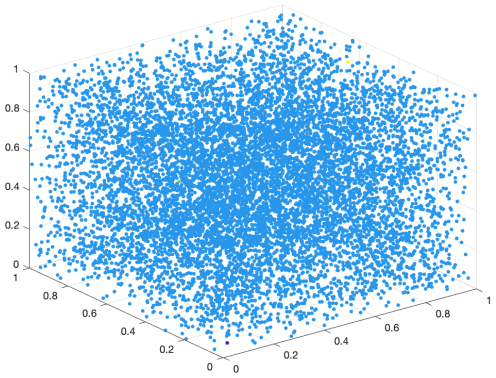

Figure 34: Single linkage applied to R^3_{10000}

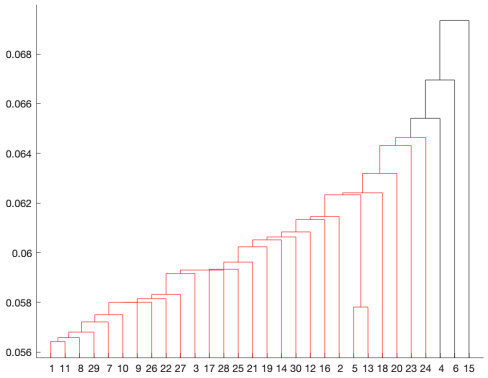

Figure 35: Dendrogram for the single linkage

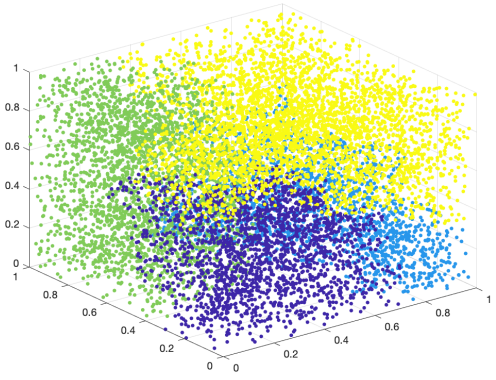

Figure 36: Complete linkage applied to R^3_{10000}

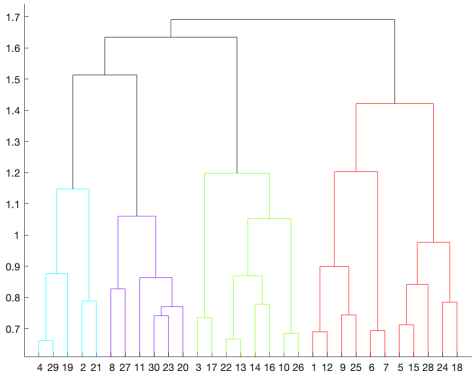

Figure 37: Dendrogram for the complete linkage

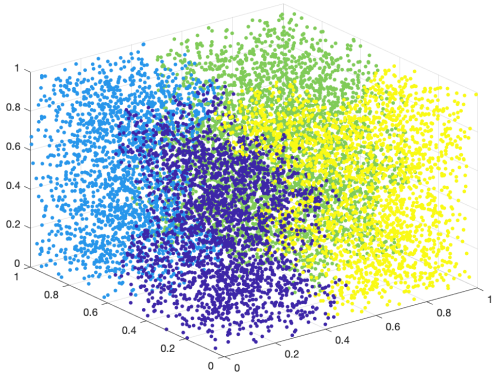

Figure 38: Average linkage applied to R^3_{10000}

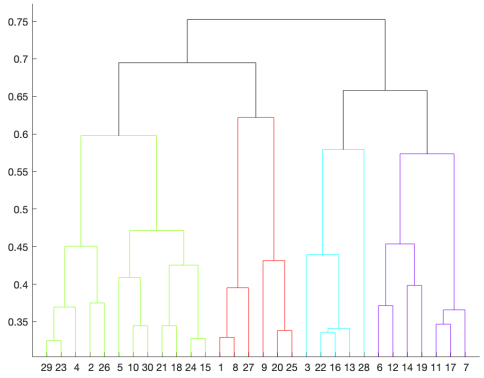

Figure 39: Dendrogram for the average linkage

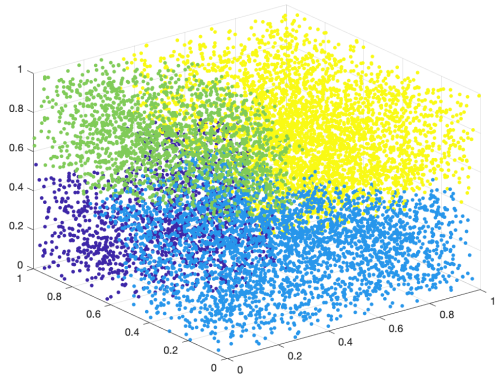

Figure 40: Ward's method applied to R^3_{10000}

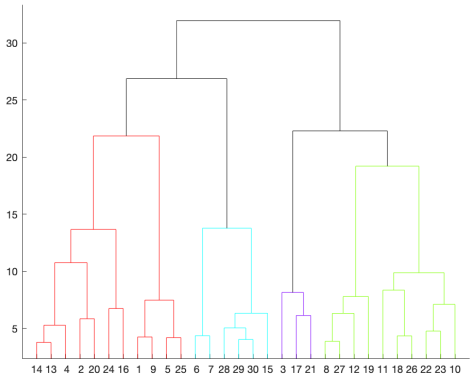

Figure 41: Dendrogram for the Ward's method

this linkage, all instances are assigned to a single cluster, even though the input parameter to the command is 4. This is reasonable if we view it along with its dendrogram, which is in Fig.35. Recall that the way we deal with the agglomerative clustering is that we first generate the dendrogram, and then we cut the dendrogram horizontally to obtain the corresponding number of clusters. Here, from the dendrogram, we can see that in effect, it is not the case that all instances are assigned to a single cluster. There is still four cluster. Three cluster, however, only contains one single instance in each. This is because we can see that if we cut the dendrogram in order to get 4 cluster, there are indeed three "subtree" that is directly its leaf, which are the three rightmost leaves. (And the big cluster is colored red, for clarification.) Therefore, the clustering result in Fig.34 is all the same color. (Just to check this deduction, I rotate the figure in Matlab, and indeed have found three different color points.) In other words, the dendrogram generated by the single linkage tends to be quite unbalanced, and seems to be very sensitive to outliers.

For the rest of the three, the algorithm behaves normally. We can see that the way complete linkage works differs from the average linkage quite obviously from the clustering results in Fig.36 and Fig.38. The color block distributed quite differently in these two plots.

Generally speaking, the dendrograms generated by the complete linkage, average linkage, and Ward's method are kind of balanced and well-behaved. Among these three, the Ward's method seems to behave the best in the way that the dendrogram generated by this method is the most balanced one. And among all three, the single linkage definitely performs the worst.

An extremely surprising result happens when we run the seemingly worst single linkage on the R^2_{1000} result, which is the unevenly distributed graph that cannot be dealt with reasonably by all of the above evaluated method. The results are shown in Fig.42-43. Although the dendrogram generated is still quite unbalanced, the cluster seems to fit perfectly with our common sense, instances belong to each block of range are assigned to the same cluster. This, however, can actually be justified by the work principle of single linkage easily. Recall that the way single linkage defines the the distance between two cluster is by the minimal pairwise distance of points from different clusters. Therefore, while choosing clusters to merge, single linkage would pick those that contain points the most closed to each other. i.e., different from the $k-$means and $k-$medoids, this linkage does not put any more weight on bigger cluster. Instead, it simply assigns instances in the way that the instances that are literally closed to each other are assigned to the same cluster. This leads to the result that this linkage can successfully deal with the uneven distribution case, even though its dendrogram is unbalanced and its performance on three-dimensional randomly generated data set is bad.

Very sadly, the other three linkages, which are able to generate relatively balanced dendrogram, all fails in the data set, just as $k-$means++, PAM, and CLARA. Interestingly but similarly, for the case L^2_{30}, for which also all previous evaluted methods fail, our single linkage seems to be able to deal with it perfectly matching our common sense as well. The results are shown in Fig.43-44. In this case, we finally find a method that can successfully cluster this data set by two separating parallel lines. This can also be explained by similar reasoning as above, considering the way single linkage defines the distance between two clusters. Note that in this case, the dendrogram is still quite unbalanced, especially in the distance between clusters (i.e., viewing from the vertical axis).

Hence, at this point, there would be no surprise for the result that single linkage is also able to cluster S^2_{400} as what we imagine the cluster should be. The results are shown in Fig.44-45.

Note that in both cases of L^2_{30} and S^2_{400}, the dendrogram generated by single linkage is quite unbalanced. It is able to, however, deal with these two cases in

the way that the cluster assignment is consistent with our common sense, which cannot be achieved by previously methods, or other linkages in agglomerative clustering. Besides, it is able to deal with uneven distributed data sets perfectly as well, successfully assigning data points belonging to the same block of range to the same cluster. Single linkage, however, performs pretty worse when dealing with randomly generated data points, i.e., data sets that are more regular. Its dendrogram is poorly generated, which leads to the result that the clusters it formed are extremely unbalanced in number of instances. On contrasts, the rest of the three linkages all work well in regular data sets. With relatively balanced and well-behaved dendrograms, the rest of the three linkages are able to cluster data sets in a reasonable and well-behaved way. Among those three, it seems that the Ward's method, which generates the most balanced dendrogram, performs the best.

On the other hand, for the agglomerative hierarchical algorithm, the complete linkage, average linkage, and Ward's method are all able to generate balanced dendrograms which lead to a relatively well-behaved and balanced clustering assignment. The single linkage, on the other hand, generates a quite unbalanced dendrogram, which results in a poorly-behaved clustering assignment for regular data set.

On contrast to its poorly generated dendrogram, single linkage seems to be the only one (among all evaluated algorithms in this paper) that is able to deal with the unevenly distributed data set and data sets that form special shapes. Because of its definition of distance between clusters, single linkage is able to assign clusters of unevenly distributed data set simply based on its density block, i.e., there is no biased (or to say "discrimination") in this algorithm, between bigger cluster and smaller cluster. This also allows it to assign clusters perfectly based on geometric shape, if the original data set forms a certain geometric shape.

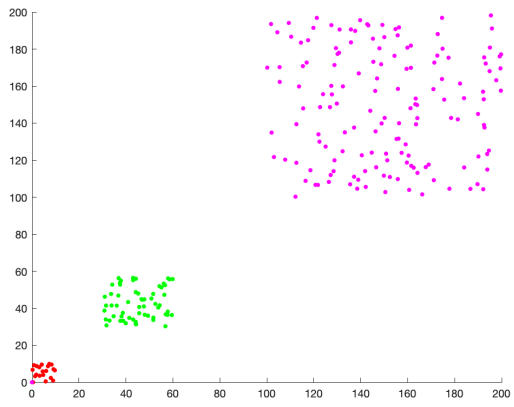

Figure 41: Single linkage applied to R^2_{1000}

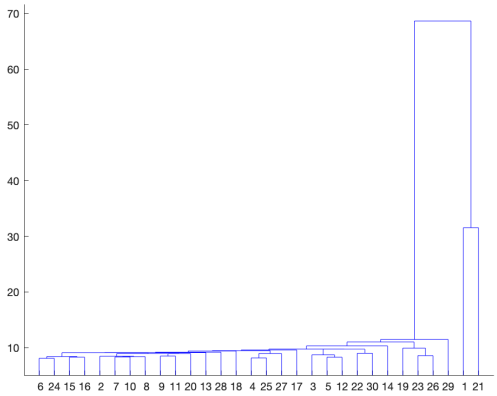

Figure 42: Dendrogram generated by single linkage for R^2_{1000}

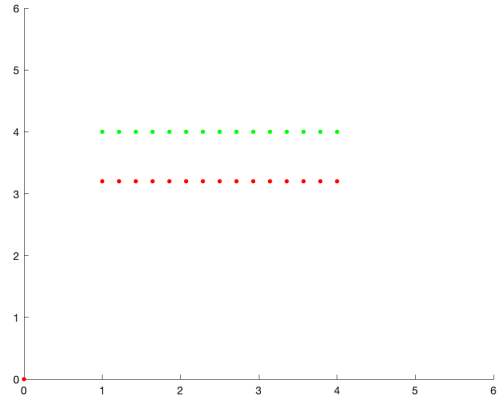

Figure 43: Single linkage applied to L^2_{30}

7 Conclusion

With multiple evaluations analyzed above, we are now ready to summarize our conclusion.

Basically, $k-$means++ is the one that is able to deal with large data set the best. By applying silhouette analysis, we are able to pick a relatively optimal k. And by running iterations with different initial cluster means, the algorithm s able to obtain average distance between all points to their corresponding cluster means converges to a minimal, which is the optimal performance and optimal choice for cluster means. Similarly, for PAM and CLARA of $k-$medoids, they perform a little bit worse in regular large data set than $k-$means++ does, while CLARA behaves slightly better than PAM. The drawback for both algorithms is that they both tend to put more weights on bigger cluster, i.e., they tend to sacrifice smaller cluster in order to get a better performance for bigger cluster. This leads to the result that these two algorithms are not good at dealing with unevenly distributed data sets. Besides, the criterion they based on to optimize prevents them from dealing with data sets that form special shapes

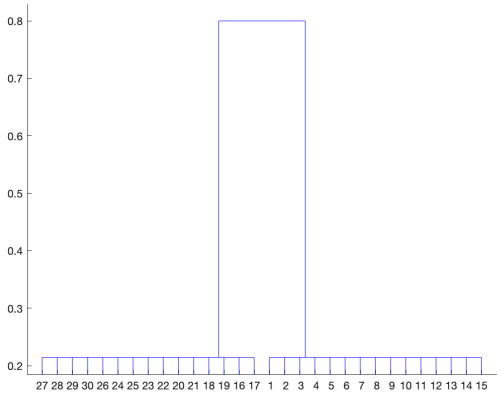

Figure 44: Dendrogram generated by single linkage for L_{30}^2

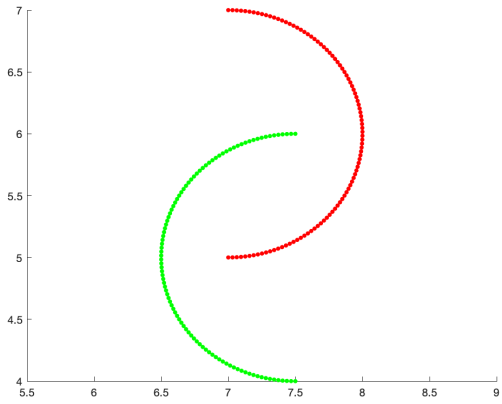

Figure 45: Single linkage applied to S_{400}^2

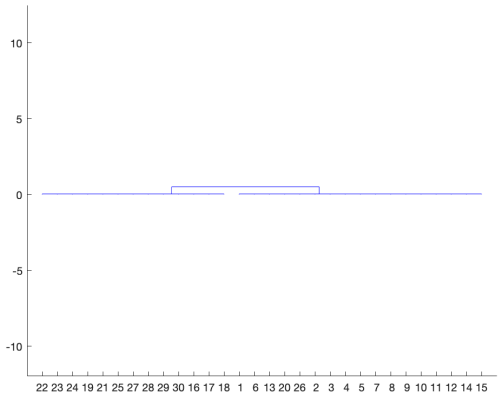

Figure 46: Dendrogram generated by single linkage for S_{400}^2

very well.

8 Discussion

At the end of this paper, a few comments are needed.

Though including discussion about the COP $k-$means, this paper did not evaluate this algorithm with experiment data sets. This is mainly because I did not find a satisfying code for this algorithm online, and I think this paper has already included enough contents, with probably no more space (both for this paper and for my energy and time) to develop that algorithm on my own. There are indeed several algorithms written by some people online, and I have also tried a few on my own. None of them, however, seems to be complete correct, and I think it is not quite useful for me to evaluate an algorithm that I cannot guarantee to be correct.

Note that this paper did not evaluate the partitional hierarchical clustering as well, and this is, still, because of my limited time and energy. I think it would be more meaningful if I can spend sufficient time and pages to discuss some algorithms in detailed, instead of going over too many algorithms so shortly and roughly.

There is another algorithm that is interesting and I intended to discuss in this paper–the constrained hierarchical clustering. Briefly speaking, this algorithm first uses partitional clustering to bisect the original data sets into multiple subset, and then applies the agglomerative clustering on each subset. In other words, it combines the partitional clustering and the agglomerative clustering. It is said that, in some papers, this algorithm performs the best among both $k-$means, $k-$medoids, and hierarchical clustering. Maybe in the future, or just this summer, I would spend more time checking this result, and if possible, I may email you my evaluation results.

Finally, note that for all my evaluations, I use my self-designed data set. I originally plan to use one or two official data sets online. Later, however, I found out that the data set I found online actually did not work as well as data sets that I designed on my own. I indeed did not design these data sets "completely arbitrarily". I design them mainly based on my understanding towards each algorithm, and design each of them in order to check some specific advantages or disadvantages of each evaluated algorithm. I acknowledge that without some official large data sets, the evaluation result in this paper is less convincing.

References

[1] Nidhi Singh,.Divakar Singh. Performance Evaluation of K-Means and Heirarichal Clustering in Terms of Accuracy and Running Time. (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 3 (3), 2012.

[2] Ying Zhao, George Karypis. Evaluation of Hierarchical Clustering Algorithms for Document

Datasets. International Conference on Information and Knowledge Management, Proceedings, 2002.

[3] Jonathan Richard Shewchuk. *Concise Machine Learning*, 21 The Singular Value Decomposition; Clustering, p.124-131, 2020.

[4] R.Mojena. Hierarchical Grouping Methods and Stopping Rules: an Evaluation. The Computer Journal, vol.20, p.359-363, 1977.

[5] Junjie Wu, Hui Xiong, Jian Chen. Adapting the Right Measures for K-means Clustering, 2009.

[6] Kiri Wagstaff, Claire Cardie. Constrained K-means Clustering with Background Knowledge. Proceedings of the Eighteenth International Conference on Machine Learning, p. 577-584, 2001.

[7] Preeti Arora, Dr. Deepali, Shipra Varshney. Analysis of K-Means and K-Medoids Algorithm For Big Data. International Conference on Information Security Privacy (ICISP2015), Nagpur, INDIA , 2015.

[8] Uğurhan Kutbay. *Partitional Clustering*, 2018.

[9] Murtagh, F., Legendre, P. Ward's Hierarchical Agglomerative Clustering Method: Which Algorithms Implement Ward's Criterion?. J Classif 31, p. 274–295, 2014.

[10] Davidson, Ian and Ravi, S. S.. *Knowledge Discovery in Databases: PKDD 2005*, Agglomerative Hierarchical Clustering with Constraints: Theoretical and Empirical Results, p. 59-70, publised by Springer Berlin Heidelberg, 2005.