ABSTRACT

A Fast Seeding Technique for k-means Algorithm

Paniz Karbasi, M.S.

Advisor: Gregory J. Hamerly, Ph.D.

The k-means algorithm is one of the most popular clustering techniques because of its speed and simplicity. This algorithm is very simple and easy to understand and implement. The first step of this algorithm is choosing k initial cluster centers. The way that this set of initial cluster centers are chosen, have a great effect on speed and quality of k-means. One of the most popular seeding techniques is k-means++ initialization, but this method needs k passes over the dataset. The Goal of this thesis is to propose a new seeding technique which chooses the initial centers much faster than k-means++.

A Fast Seeding Technique for k-means Algorithm

by

Paniz Karbasi, B.S.

A Thesis

Approved by the Department of Computer Science

_____

Gregory D. Speegle, Ph.D., Chairperson

Submitted to the Graduate Faculty of
Baylor University in Partial Fulfillment of the
Requirements for the Degree of

Master of Science

Approved by the Thesis Committee

_____

Gregory J. Hamerly, Ph.D., Chairperson

_____

Gregory D. Speegle, Ph.D.

_____

Keith E. Schubert, Ph.D.

Accepted by the Graduate School
December 2014

_____

J. Larry Lyon, Ph.D., Dean

Page bearing signatures is kept on file in the Graduate School.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to express my special appreciation to my advisor Dr. Greg Hamerly who has been a tremendous mentor for me. Thank you for teaching me how to do research. I owe you my sincerest gratitude.

I would also like to thank the members of my committee, Dr. Greg Speegle, and Dr. Keith Schubert for their time and patience.

To my grandparents, Nani and Pedar: Thank you for teaching me to be patient and assiduous when moving toward my goals.

To my parents, Shiva and Hamid: Thank you for teaching me to be strong and hopeful when I face the difficulties in my life.

To My Husband, Bahram

The light and love of my life, without whom none of this would have been possible

CHAPTER ONE

Introduction

1.1 Goal of the Thesis

The main goal of this thesis is to review some of the most popular seeding techniques for the k-means algorithm, and then develop a new and fast initialization seeding method for choosing the initial cluster centers, and at the end, compare the proposed method in this thesis with two of the widely used seeding techniques for the k-means algorithm.

1.2 A Brief Overview of k-means

The k-means algorithm, is one of the oldest and most popular algorithms for data clustering, since it is fast, easy to understand, and can be implemented simply. It was originally developed and used for the task of vector quantization, but it has been used in a wide variety of applications (Jain 2010). Having the data and k as input arguments, k-means groups the data into k clusters. Table 1.1 contains the most important and frequently used terms used in this thesis to explain the algorithms, mathematical definitions, and results presented.

When tracing back k-means to its origins, we see that this algorithm has been proposed by several scientists in different forms and under different assumptions. In (Bock 2007) some historical issues related to the k-means algorithm have been discussed, and it has been mentioned that the k-means clustering problem was first mentioned in 1950, but not with the same name. Therefore, it is an old problem, but still very popular.

This chapter starts with describing some of the most important applications of the k-means algorithm in different areas. After introducing the k-means algorithm, we continue with discussing the math behind the general k-means problem. Next in

this chapter, we discuss different factors that affect the clustering results in terms of the speed and accuracy. After discussing the factors that affect the clustering results, the first step of the k-means algorithm, which is the motivation for this research is discussed in more details. Finally the outline for the rest of the thesis is described at the end of this chapter.

Table 1.1. Frequently used terms

Name	Domain	Description
n	$\mathbb{N}$	Number of data points to cluster
d	$\mathbb{N}$	Dimension of data points to cluster and cluster centers
k	$\mathbb{N}$	Number of cluster centers (also number of clusters)
a	$\mathbb{N} \to \mathbb{K}$	Index of assigned center for each data point, $\mathbb{K} = \{1, 2, \ldots, k\}$
X, Dataset	$\mathbb{R}^{n \times d}$	Set of data points to cluster, indexed as $x^{(i)}$ for $i \in \{1...n\}$
Cluster	$\mathbb{R}^{p \times d}$	A cluster is a set of $0 \leq p \leq n$ points such that any point in a cluster is closer to every other point in the cluster th -an to any point not in the cluster.
C	$\cup_{j=1}^{k} c(j)$	Set of clusters, indexed as $c(j)$ for $j \in \{1...k\}$
μ, Centroid	$\mathbb{R}^{k \times d}$	Set of cluster centroids, indexed as $\mu_c(i)$ for $i \in \{1...n\}$. $\mu_c(i)$ indicates the mean of the cluster c that $x^{(i)}$ belongs to it[1].
J, SSE	$\mathbb{R}^{+}$	Distortion, Sum of the squared errors
Ext	X	An arbitrary extreme point chosen from X

[1]The difference between C and μ is that each $c_j \in C$ is a cluster that contains some data points, while μ_{c_j} is the mean of the data points in the cluster $c_j \in C$.

1.3 Applications of k-means

The k-means algorithm is used in many applications in a wide range of areas, since it is fast and easy to implement. Some of the most popular applications are discussed below.

1.3.1 Cluster Analysis

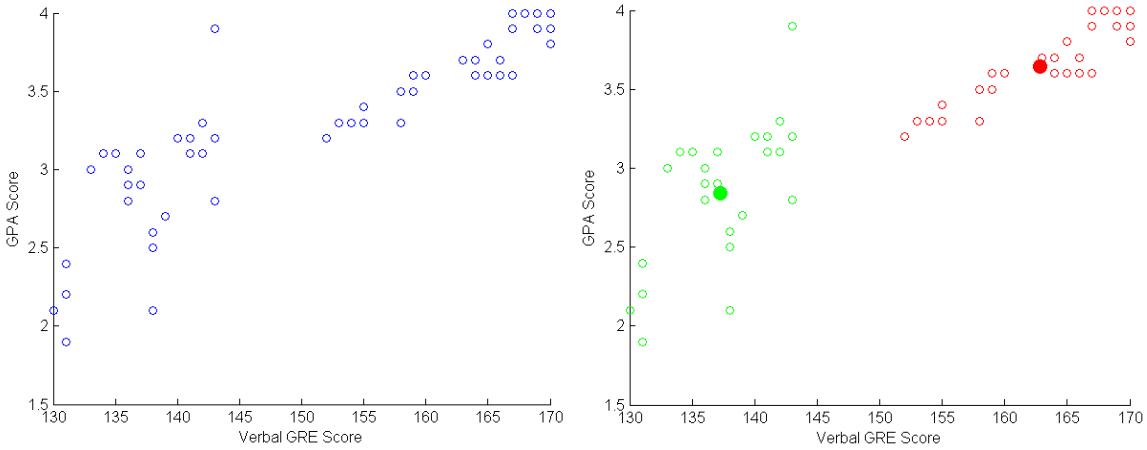
In cluster analysis, the k-means algorithm can be used to partition the input data set into k partitions (clusters). We will describe a naive example that shows the application of k-means in cluster analysis:

An imaginary business department has a large applicant database which has thousands of graduate applicants with data on the GRE, GPA, country of citizenship, and other attributes such as age, sex, and etc.. The department wants to perform a survey to know the characteristics (e.g. country of citizenship, GRE and GPA scores) of the applicants who have similar GRE and GPA scores, and thus to start a collaboration (possibly an exchange student program) with the universities in the countries that their citizens have higher scores in GPA and GRE. Figure 1.1 shows the plot of verbal GRE score vs. GPA score of a sample of applicants.

If we perform k-means with $k = 2$, on the data illustrated in Figure 1.1a, one possible output of the k-means clustering could be the two clusters that are illustrated in Figure 1.1b. After running k-means on the dataset, since two clusters are generated, we can find the average GRE and GPA score in each cluster, and find the country of citizenship of those applicants who belong to the group with the higher average GRE and GPA score.

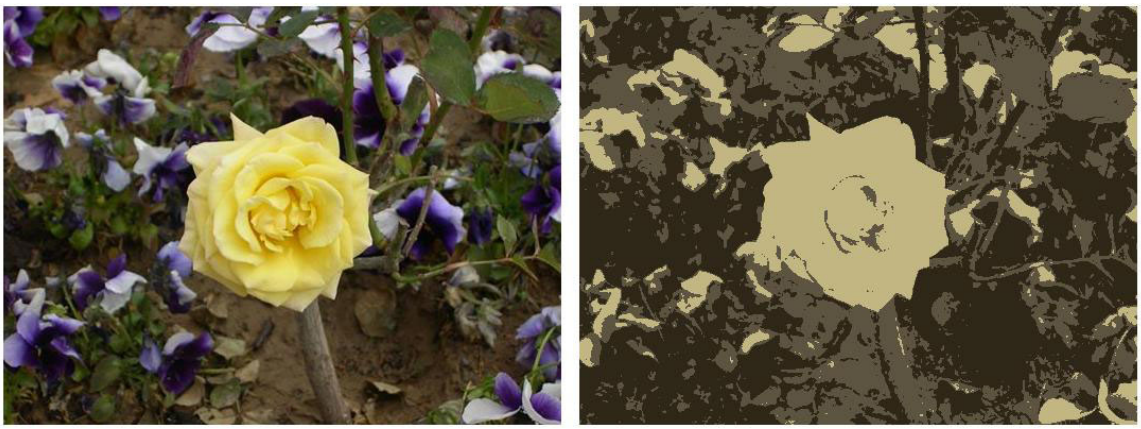
1.3.2 Data Compression

One of the methods used for image compression, is called color quantization. Color quantization is a process that reduces the number of distinct colors in an image to a fixed number of colors, k (Celebi 2011). See Figures 1.2a, and 1.2b as examples.



(a) GRE and GPA grades of 50 applicants (b) Clustered data after using k-means

Figure 1.1. (a) A synthetic dataset of 50 data points in 2-dimensional space. The data points represent the GRE and GPA grades of 50 applicants. (b) Clustering result after using k-means with $k = 2$



(a) Original image (b) Quantized image

Figure 1.2. (a) Original image (b) Quantized image after using k-means with $k = 3$

1.3.3 Computer Vision

One of the most important issues in computer vision field is that of image analyzing and understanding. Image segmentation is the process of partitioning a digital image into multiple segments (sets of pixels), which can be further analyzed.

This is typically used to identify objects or other relevant information in digital images. The main goal of image segmentation is to modify the representation of an image into something that can be analyzed easier. Image segmentation have many applications in different areas such as content-based image retrieval, medical imaging, object detection, etc. (Chitade and Katiyar 2010).

1.3.4 More Applications

We discussed the general applications of the k-means algorithm above. In addition to the general applications of k-means, there are much more specific applications in different areas, which a few of them are described below.

- Finding a good initialization for a more costly learning method (Bottou and Bengio 1995)
- Unsupervised feature learning in single-layer neural networks (Coates, Ng, and Lee 2011)
- Enabling lossy compression of audio or image data by selecting the codewords for vector quantization (Linde, Buzo, and Gray 1980)
- Content based image retrieval using k-means clustering technique (Murthy, Vamsidhar, Kumar, and Rao 2010)

Also, in unsupervised learning unlike the supervised learning, the task is to give labels to objects without knowing the true labels. k-means, as a clustering algorithm that partitions a dataset into several groups is naturally performing a classification task, where the labels are unknown.

In the next section we discuss some mathematical definitions related to the k-means algorithm that help better understand what is happening inside this fast and simple algorithm. Moreover, we talk about the factors that affect the clustering results of this algorithm, and conduct some experiments to show the importance of the k-means initialization.

1.4 The First Step of k-means, a Critical Step

When clustering with k-means, the goal is to minimize a certain mathematical criterion which is called the distortion (J), between the data points and their assigned centers. The distortion can be improved in two ways; (1) by changing the data point's current clusters (assigning the data points to different clusters), and (2) by moving the cluster centers. In particular, given a set of clusters (C) which include the data points, and a set of cluster centers (μ), the distortion function of the k-means algorithm is defined as:

$$J(C, \mu) = \sum_{i=1}^{n} ||x^{(i)} - \mu_{c \in C}(i)||^2 \tag{1.1}$$

In particular, J measures the sum of squared distances between each training example $x^{(i)}$ and the cluster centroid $\mu_c(i)$ to which it has been assigned. In literature, besides the distortion there are other names for the above function such as potential or SSE.

Various heuristics have been developed to provide approximate solutions to the k-means problem (Tarsitano 2003). Among these methods Lloyd's algorithm (Lloyd 1982) is the simplest and most commonly used. This algorithm, which is basically known as the standard k-means algorithm, minimizes the distortion function in three basic steps; (1) initialization step, (2) assignment step, and (3) update step. Lloyd's algorithm is a gradient descent heuristic algorithm for minimizing k-means distortion function. The more details of this algorithm is described in Algorithm 1. Here is a brief description of the k-means algorithm:

1. Initialize cluster centroids $\mu_{c(1)}, \mu_{c(2)}, ...\mu_{c(k)} \in \mathbb{R}^d$
2. Repeat until convergence: {

   Assignment step: Assign each data point to the closest cluster center

   Update step: Update the mean of each cluster

}

Algorithm 1 Lloyd's k-means algorithm

```
1: procedure LLOYD(X,C)
2:     while not converged do
3:         for each i ∈ n do
4:             a(i) ← 1
5:             for each c ∈ C do
6:                 if ‖x^(i) − μ_c(i)‖ < ‖x^(i) − μ_c(a(i))‖ then
7:                     a(i) ← c
8:         for each c ∈ C do
9:             μ_c(i) ← mean of {x^(i) | a(i) = c}
```

k-means is guaranteed to converge, since the inner-loop of k-means repeatedly minimizes J with respect to c, while holding μ fixed, and then minimizes J with respect to μ while holding c fixed. Therefore, J monotonically decreases until there is no more change in the clusters. Also, there is a finite number of ways to partition the n data points among k clusters (Bottou and Bengio 1995).

The number of ways that a set of n data points can be partitioned into k non-empty clusters, is given by Stirling numbers of the second kind (Graham and Knuth 1989):

$$S(n,k) = \frac{1}{k!}\sum_{i=0}^{k}(-1)^{k-i}\begin{pmatrix} n \\ i \end{pmatrix} i^n \tag{1.2}$$

Based on the Equation (1.2), it can be seen that enumerating all possible clusterings to determine the global minimum of (1.1) is computationally prohibitive except for very small datasets (Kaufman and Rousseeuw 2009). In fact, minimizing the distortion function is known to be NP-hard for the general problem of partitioning d-dimensional data into k clusters (Aloise, Deshpande, Hansen, and Popat 2009, Mahajan, Nimbhorkar, and Varadarajan 2009). Based on this fact, k-means seeks local minimum solutions rather than the global. But, the k-means algorithm can get stuck at poor solutions. When we speak of quality of solution of k-means, we consider a solution that better minimizes the distortion function (with a fixed k) to be a better-quality clustering. For instance, Figures 1.3a, 1.3b, 1.3c, and 1.3d show

two different clusterings of the same data (when $k = 25$) while one solution has a lower distortion value, or in other words better minimizes the distortion function, and has a higher-quality solution.

Although the k-means algorithm is being used widely, there are still some drawbacks with this algorithm (Pena, Lozano, and Larranaga 1999):

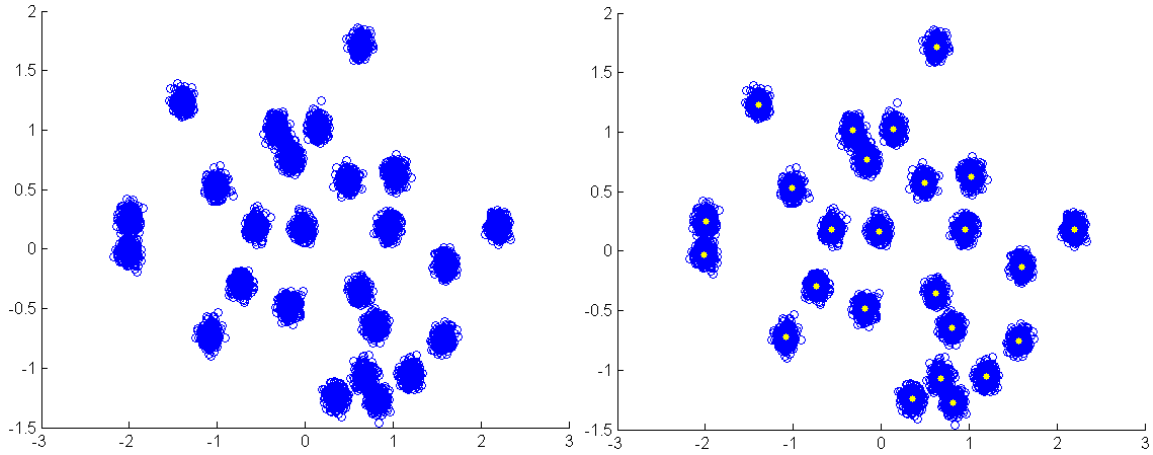
- Initial cluster centers determine the final structure of the clustering.
- Outliers may have a disproportionate impact on the final cluster configuration. This can result in the data points that are masked by the clustering, but actually should be declared outliers.
- The k-means algorithm assumes that the number of clusters K is known beforehand which, is not necessarily true in real-world applications.

Among the above factors, the focus of this research is on the k-means initialization, and the goal of this thesis is to propose a new method for the k-means initialization, that results in better qualified clustering(s) in fewer iterations.

With a fixed k, k-means convergence to a good or poor quality solution is heavily dependent on the initial cluster centers. Also, the number of iterations of the k-means is affected by the initialization of this algorithm. In general, every value of k has at least one clustering with minimum distortion (it is possible that different sets of initial cluster centers, generate solutions or clusterings with the same distortion). The closer the initial cluster centers to the means of an optimal clustering, the better results we get with respect to number of iterations and distortion. To show the relation of the distortion, and number of iterations with the initialization of the k-means, we can run k-means on a dataset and verify the results of this algorithm with different sets of initial cluster centers:

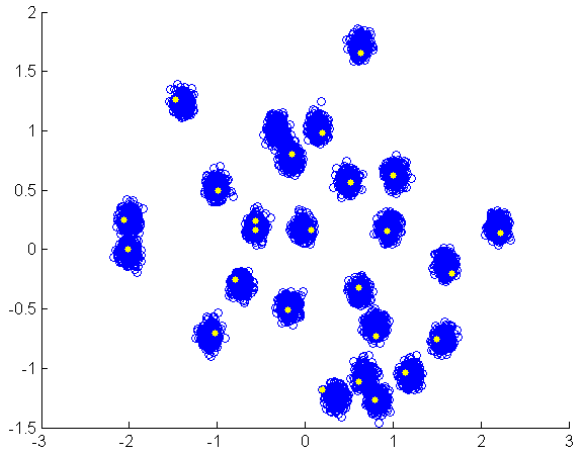
We have created a synthetic dataset, with $n = 10000$, $d = 2$, and 25 clusters (Figures 1.3a - 1.3d). We also have the ground truth file for this dataset which contains means of the clusters. The data points are in blue, and each big blue circle

illustrates a group of data points which belong to the same cluster. Clusters have the same size (each blue circle or cluster contains 400 data points). Also, the yellow dots represent the initial cluster centers.

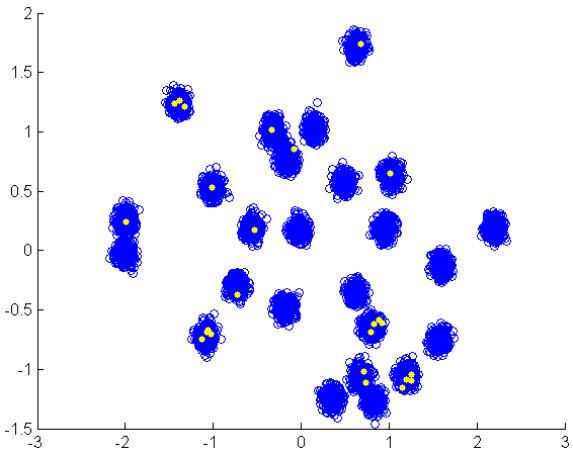


(a) Original Data

(b) Optimal clustering



(c) A good initialization



(d) A bad initialization

Figure 1.3. Illustration of various k-means solutions with $k = 25$. (a) is the original data which is composed of 25 well-separated clusters, with $n = 10000$, $d = 2$. (b) illustrates the optimal solution to the k-means with $k = 25$. Note that the yellow dots in this sub-figure, are both the initial centers and final centers. (c) illustrates the initial cluster centers which are close to the optimal solution in Figure 1.3b, since the final distortion is very close to the final distortion of the optimal clustering, we also could call it a good clustering. (d) illustrates the initial cluster centers which are not close to the optimal solution, since the final distortion is not close to the final distortion of the optimal clustering, we also could call it a bad clustering

In Figures 1.3c, and 1.3d, two different sets of initial cluster centers have been used, and the Lloyd's k-means algorithm is run on the dataset with $k = 25$. More specifically, in Figure 1.3c, a set of initial cluster centers very close to the centers of the clusters in the optimal clustering have been used, while, in Figure 1.3d, some different initial cluster centers that are chosen uniformly at random have been used.

We have selected two different sets of initialization points which are illustrated in Figures 1.3c, and 1.3d. Having these datasets with different centroids, we have performed k-means on them. Table 1.2 shows the distortion before and after k-means convergence, and the number of iterations with two different initializations.

Table 1.2. Clustering results on the synthetic dataset illustrated in Figures 1.3b, 1.3c, and 1.3d, with three different initializations.

Initialization	Initial distortion	Final distortion	# of iterations
Optimal Clustering	51.11	51.11	0
Well-chosen centers	70.31	67.72	12
Random chosen centers	411.552	315.655	35

In Table 1.2, the well-chosen initial cluster centers, are the centers illustrated in Figure 1.3c, and uniformly random chosen initial centers are illustrated in Figure 1.3d. Note that the well-chosen cluster centers are closer to the optimal clustering (Figure 1.3b), even if a few of them are placed within the same clusters. While in the case of uniformly random chosen data, there exist more initial centers that are placed within the same clusters. As you see in Table 1.2, in the case that initial cluster centers are very close to the actual means of the clusters, the number of iterations is much fewer than the case that the initial cluster centers are chosen uniformly at random. Also, distortion, when centers are chosen close to the means of the clusters, is much better than the case that centers are chosen uniformly at random. This short and simple example shows how heavily k-means is dependent on its initialization.

1.5 Outline

In the second chapter, we discuss some of the widely used seeding techniques with their negative and positive aspects. In chapter three, we introduces a new initialization technique for k-means algorithm. In chapter four, we compare the result of k-means augmented with different initialization methods in different metrics, and finally in the last chapter, we draw some conclusions based on the results of chapter four.

CHAPTER TWO

Related Work

The k-means algorithm is highly sensitive to the initial placement of the cluster centers due to its gradient descent nature. Since the initialization of the k-means, affects the results of the final clustering, there have been a wide range of different techniques that are used for selecting the initial cluster centers (Celebi, Kingravi, and Vela 2013). In this chapter, some of the most important and commonly used initialization techniques for the k-means clustering are described with their positive and negative aspects. These methods are considered to be heuristic, because they are some techniques for finding a clustering which is not guaranteed to be optimal.

2.1 Random Partition Method

In the random partition method, we divide the dataset into k clusters. In other words, a cluster is randomly assigned to each data point, and then we proceed to the update phase of the k-means algorithm. This method is fast and has linear time-complexity. The main drawback of this method is that in a clustered dataset, it is possible that the data points that belong to the same true cluster, be placed in the different clusters after using this method.

2.2 Forgy's Method

Forgy's method (Pena, Lozano, and Larranaga 1999) chooses k instances from the dataset uniformly at random, and assigns the rest of the data points to the closest instance. This method is very applicable, because of its simplicity and high speed. The rationale behind this way of initialization is that random selection is likely to pick data points from dense regions (data points that are good candidates). A major problem with Forgy's method is that if an outlier is chosen as an initial cluster center, it is possible that no other data point is assigned to it, and therefore the cluster with

the outlier[1] as its center remains empty. Another drawback of this method is that there is no mechanism to avoid choosing data points that are too close to each other. Actually, running Forgy's method several times, and picking the best solution (the one that most minimizes the distortion) is one of ways that is widely used for improving the k-means solution.

The difference between Forgy's method and Random Partition method is that the first one aims at spreading centers out in the data, while the second method tends to place the centers in a small area near the middle of the dataset (Hamerly and Elkan 2002).

2.3 Kaufman's Method

In Kaufman's method (Kaufman and Rousseeuw 2009), the most centrally located object is selected as the first centroid. The next center is chosen to be the point that most reduces the SSE. We continue selecting points in the same way until we get k points. The time complexity of this method is $O(n^2)$, since pairwise distances between the data points need to be calculated in each iteration. If this algorithm is to be considered useful for large datasets a subsample of the data points must be used instead when finding the seeds (Redmond and Heneghan 2007). The details of the Kaufman's initialization method are described in Algorithm 2.

2.4 Furthest-First Method

In the furthest-first method (Hochbaum and Shmoys 1985), the first center is chosen uniformly at random from the dataset. Next, the point which is farthest from the first center is chosen. Third center is the point that is farthest from both previous centers. We continue selecting points in this way until there are k centers. In the next page we describe the math behind the furthest-first algorithm in more details.

[1]An outlier is an observation of the data that deviates from other observations so much that it arouses suspicions that it was generated by a different mechanism from the most part of data (Williams, Hawkins, Gu, Baxter, and He 2002).

Algorithm 2 Kaufman's algorithm

1: **procedure** KF(X,K)
2: $c_1 \leftarrow$ Select the most centrally located data point
3: **for each** non-selected $x^{(i)} \in X$ **do**
4: **for each** non-selected $x^{(i')} \in X$ **do**
5: $d_{i'i} \leftarrow \|x^{(i)} - x^{(i')}\|$
6: $D_{i'} \leftarrow min_s d_{si'}$ ▷ Being s one of the selected seeds
7: $C_{i'i} = max(D_{i'} - d_{i'i}, 0)$
8: $G \leftarrow \sum_{i'} C_{i'i}$ ▷ The gain of selecting $x^{(i)}$
9: $c_j \leftarrow$ Select $x^{(i)}$ which maximizes $\sum_{i'} C_{i'i}$
10: **if** K seeds are selected **then**
11: Stop
12: **else**
13: go to line 2

Based on the furthest-first algorithm, the next center is the point that satisfies the following statement:

$$argmax_{x \in X} min_{c \in C} d(x, c) \tag{2.1}$$

In the above statement, $d(x, c)$ is the distance between a data point and a cluster center. This gives a way to perform a semi-random initialization that attempts to pick initial means as far from each other as possible. In this heuristic, the only randomness is the selection of the first center. After that, it is completely deterministic (except in the rare case that when choosing the next center, there are multiple equidistant data points).

The drawback of this method is that it tends to select outliers, which are not good candidates for being cluster centers. When the first seed is selected uniformly at random from dataset, the next seed is the data point which is furthest from the first seed, which could be an outlier. Also, when selecting the remaining centers, there is still a great chance that we choose outliers, because still we look for the data points that are furthest from the previously chosen centers. The good thing about this method is that it spreads out the centers throughout the dataset, since it selects

the data points which are furthest from the previously chosen centers. Spreading out the centers is good in general, because it makes it less probable that data points that are too far from each other, be at the same cluster (we want close data points be at the same cluster). Also, if the initial centers are too close to each other, it makes the initial distortion greater in comparison with the case that they are far from one another, and in general, we are interested that k-means algorithm starts with a lower distortion. Moreover, with the centers spread out throughout the dataset, there is less chance of splitting clusters and with the centers being a better starting point, k-means can converge with fewer iterations. By adding a bit more randomness to the heuristic, it can be changed to an algorithm called k-means++ which is described in the next section.

2.5 k-means++ Initialization

k-means++ is one of the most popular seeding techniques because it makes k-means converge faster (with reducing the number of iterations) to a better distortion. Also, this algorithm is proved to be $O(\log k)$-competitive with the optimal clustering (Arthur and Vassilvitskii 2007). This does not tell that it will reach the global optimum, but it does tell that it will get reasonably close to the global optimum. In particular, if ϕ is the value obtained by running k-means++, then this will not be too far from optimal distortion (ϕ^{opt}). More precisely,

$$E[\phi] < 8(\log(k) + 2)\phi^{opt} \tag{2.2}$$

Let $D(x)$ denote the shortest distance from a data point to the closest center we have already chosen. Then, Algorithm 3 describes the steps of the k-means++ initialization in detail. Note that k-means++ consists of two parts: initialization (choosing initial cluster centers), and k-means algorithm itself. Algorithm 3 describes just the k-means++ initialization. It is obvious that after choosing the k initial centers we proceed as with the standard k-means algorithm.

Algorithm 3 k-means++ Initialization

1: **procedure** KPP(X,K)
2: $C \leftarrow$ Choose one center uniformly at random from X
3: **while** $|C| < k$ **do**
4: $D(x) \leftarrow$ Shortest distance from $x \in X$ to the closest center in C
5: $c_i \leftarrow$ Choose $x \in X$ with probability $\frac{D(x)^2}{\sum_{x \in X} D(x)^2}$
6: $C \leftarrow C \cup \{c_i\}$

The authors of k-means++ have proposed a specific way of choosing the initial cluster centers for the k-means algorithm: the first cluster center is chosen uniformly at random from the dataset, after which each subsequent cluster center is chosen from the remaining data points with probability proportional to its squared distance from the closest center to which it has already been assigned. The results of comparing k-means++ and k-means augmented with Forgy's method have shown that using k-means++ in most of the cases reduces the number of iterations of the k-means algorithm, and also converges to a better local minimum (Arthur and Vassilvitskii 2007). Having fewer number of iterations and less runtime is critical when using k-means for clustering, especially when it comes to the big datasets. If the dataset is too big, each iteration of the k-means algorithm could be really slow, therefore having a method which causes k-means converge faster, could be really beneficial.

What is k-means++ doing? The first center is chosen uniformly at random from the dataset. The second center is chosen to be, on average, far from the first. The third center is chosen to be far from both the previous centers, and so on. This way of seeding is similar to the Furthest-First method. The difference is that instead of selecting the data points that are furthest from the previous centers, it chooses the next seed based on a probability proportional to its squared distance from the closest center to which it has already been assigned. The intuition that spreading out the k initial cluster centers is a good thing is behind this approach. Thus, with the centers that are not too close to each other, we can avoid splitting clusters in the dataset.

On the other hand, this way of initialization is as costly as one iteration of the k-means algorithm. In other words, k-means++ needs k passes over the data points in order to find the k initial cluster centers. Moreover, k-means++ is not scalable. Because of k-means++ inherent sequential nature which limits its applicability to massive data. Recently a parallel version of this seeding technique has been introduced. The authors of k-means++ in (Bahmani, Moseley, Vattani, Kumar, and Vassilvitskii 2012) have shown how to reduce the number of passes needed to obtain, in parallel, a good initialization. They have proved that the modified version of k-means++ reaches a nearly optimal solution after a logarithmic number of passes, and then shown that in practice a constant number of passes suffices.

The next chapter introduces a new method which selects the initial cluster centers much faster than k-means++. The results of comparing the proposed method, k-means++, and Forgy's method have been shown in chapter four.

CHAPTER THREE

Methodology

3.1 A New Initialization Method for k-means

In this chapter, we introduce a new method for finding k initial cluster centers for the k-means algorithm. The proposed method, basically finds the initial cluster centers based on their distance to an arbitrary extreme point in the dataset. First, we explain the algorithm with a naive example and then we illustrate the algorithm more deeply.

3.1.1 High-level Overview of the Proposed Method

Before starting the high-level overview of the proposed method, we need to clarify a mathematical concept that we have used in our algorithm, "extreme point". Given a dataset X, an extreme point, in our case, is an arbitrary data point such that it is farthest from another arbitrary data point in X.

Consider a dataset having 10 data points in 2-dimensional space. The goal is run the k-means algorithm to find k clusters of data points. Figure 3.1 shows a sample dataset in 2-dimensional space:

Figure 3.1. A sample dataset

First, select an arbitrary extreme point from the dataset (the extreme point is shown in red in Figure 3.2). Next, calculate the distance of the remaining points with respect to the extreme point. This step is illustrated in Figure 3.3. Then, sort the

points based on their distance from the extreme point. Once the distances are sorted, starting from the maximum distance, calculate the difference of each two consecutive distances:

Figure 3.2. An extreme point is chosen from the dataset

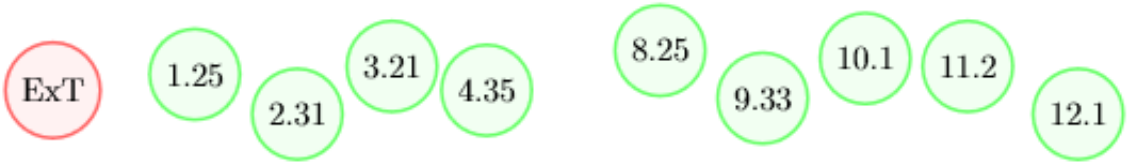


Figure 3.3. Calculating the distances with respect to the extreme point

$12.1 - 11.2 = 0.9$, $11.2 - 10.1 = 1.1$, $10.1 - 9.33 = 0.77$, $9.33 - 8.25 = 1.08$ ...

After calculating the difference of each two consecutive distances, calculate the average of the values from the previous step:

$$average = 1.35$$

Other than the way that we discussed above, there is an alternative way for calculating the average of the difference between each two consecutive distances which is discussed here; instead of calculating the difference of each two consecutive distances and dividing that by the number of the distances, we can calculate the difference between the first and the last distance, and then divide the difference by the number of distances minus one. Based on this method, the average of the distances of Figure 3.3 is calculated in the following way:

$$average = \frac{12.1-1.25}{8} = 1.35$$

Next, partition the distances into m groups such that the difference between each two consecutive distances in a group is less than or equal to the average value from the previous step (Figure 3.4). Note that before partitioning the distances, we do not know the value of m. The value of m is discovered after partitioning.

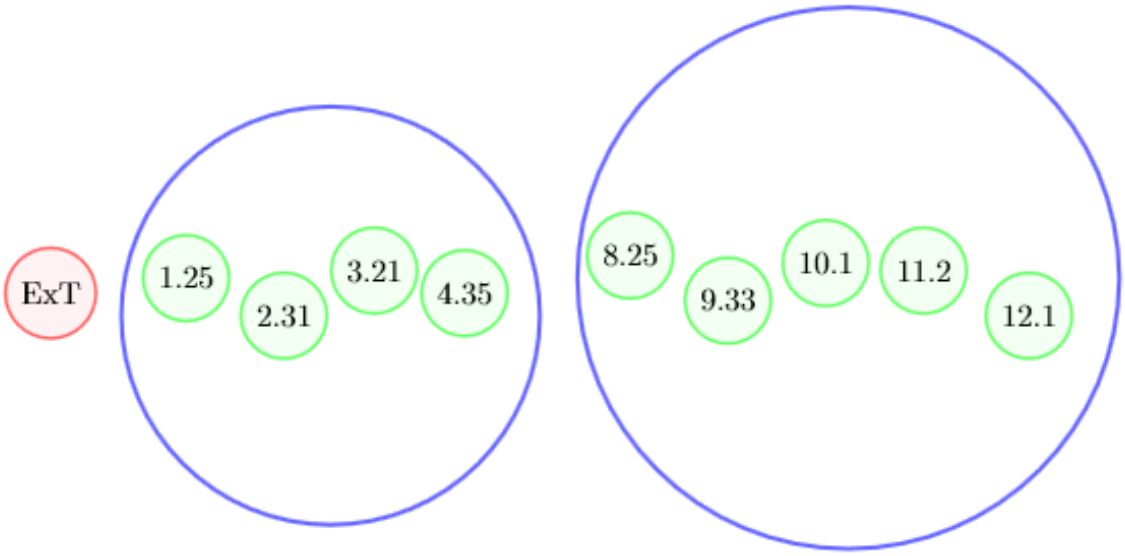


Figure 3.4. Grouping the data points

At this point, there are $m = 2$ groups of points. From now on, based on k and m proceed with one the following cases:

- If $k < m$, merge the groups such that after merging, there exist k groups. Then pick one point from each of m groups.
- If $k = m$, pick one point from each of m groups.
- If $k > m$, pick points from the groups in a round-robin fashion until we have k points.

For example, if $k = 5$ ($k > m$), starting from the furthest group of points, one point is selected from each of the groups at a time. When reached to the last group, again we start from the first group and pick one point at a time until we get 5 points. Note than there may be some groups with just one point, like in our example (there

are two groups of one point). In this case, if we had to pick more than one point from the groups, we simply ignore the groups of one member that we have chosen a point from before and, proceed to the next groups. If $k = 2$ ($k = m$), simply pick one point from each of m groups.

3.1.2 Pseudo-code of the Proposed Method

In this section, we describe the steps of the proposed method more in depth, and further analyze its time complexity. Algorithm 4 shows the steps of the proposed method.

Algorithm 4 Proposed method

```
1:  procedure SEEDING(X,K)
2:      ext ← choose an arbitrary extreme point from X
3:      for i ∈ N do
4:          dist(i) ← ||x^(i) − ext||
5:      sort(dist)                          ▷ Sort the distances from step 4
6:      for i ∈ N − 1 do
7:          diff(i) ← |dist(i) - dist(i+1)|
8:      avg ← average(diff)                 ▷ Calculate the average of values in step 7
9:      G ← partition(dist,avg)             ▷ Divide the distance list into m groups
10:     m ← |G|
11:     if m > k then
12:         merge(G)
13:     else if m < k then
14:         groupSeeding(G)
15:     else
16:         for each g ∈ G do
17:             pick the most centrally located value in g
18:     retrieve the k original data points
```

There are some methods used in Algorithm 2 pseudo-code (e.g. *merge()*). In the following subsections, we describe the behavior of these functions.

3.1.2.1 partition: line 9. At this point, we have a sorted list containing the distances of the data points with respect to the extreme point and the average

value of the differences of each two consecutive distances. We divide the sorted list of distances into m groups such that the difference of each two consecutive distances in a group, is less than or equal to the average value.

3.1.2.2 merge: line 12. After partitioning the sorted list of distances into m groups, if $m > k$, we need to merge the groups into k groups. For this, divide m by k, or $q \leftarrow \lfloor \frac{m}{k} \rfloor$, and r is the remainder of this division. If $q > 1$, we merge the m groups into $k - 1$ super-groups each containing q groups, and one super-group containing $r + q$ groups. Next based on lines $16 - 18$ of Algorithm 4, we pick k values from the middle of the super-groups (e.g if the size of a super-group is 10, the value which we select is the 5th value in the super-group. If the size of a super-group is 11, the value we select is either the 5th or 6th element). At the end, we retrieve the original data points based on the selected distances. For illustrating this step consider the following example which is the sorted list of the distances of the data points with respect to the extreme point (assuming the size of dataset is 20).

$1, 2, 3, 20, 21, 22, 50, 51, 52, 70, 72, 73, 91, 94, 95, 115, 116, 118, 122$

The average value of the differences of each two consecutive distances is 6.72. Therefore, we can divide the above list into the following groups:

$[1, 2, 3], [20, 21, 22], [50, 51, 52], [70, 72, 73], [91, 94, 95], [115, 116, 118, 122]$

At this point, we know that we have $m = 6$ groups. If $k = 3$, then $q = 2$ and $r = 0$. Therefore, we have to merge the groups into 2 super-groups each containing 2 groups, and one super-group containing $0 + 2$ $(r + q)$ groups. Note that groups are also sorted internally and externally.

$\{[1, 2, 3], [20, 21, 22]\}, \{[50, 51, 52], [70, 72, 73]\}, \{[91, 94, 95], [115, 116, 118, 122]\}$

Then, we pick 3 values at the middle of each super-group: 3 from the first super-group, 52 from the next super-group, and 115 from the last super-group. At the end, we retrieve the original data points from the dataset based on the selected distances.

Now, we consider the second case: if $q = 1$. In this case, we first pick k groups randomly, next, we pick values that are stored in the middle of each group. Back to our example, if $k = 4$, then $q = 1$. Therefore, we pick 4 groups randomly, and select the values which are in the middle of each group.

3.1.2.3 groupSeeding: line 14. In this subsection we consider the case that the number of groups is less than k. If $m < k$, we need to select more than one value form some of the groups. For this, divide k by m, or $q \leftarrow \lfloor \frac{k}{m} \rfloor$, and r is the remainder of this division. In other words, $k = q \times m + r$. In this case, we need to pick $q + 1$ values from r groups, and q values from $m - r$ groups.

Also, there is second way of choosing values from groups when $m < k$. This way is more like a round-robin fashion. First, we select a value from the first group, next from the second group, until we get to the last group. Again, starting from the first group, we select values until we get k of them.

3.1.3 Extreme Point Selection

Choosing an extreme point from the dataset, is the first step in the proposed method. Basically, a simple and straightforward way for choosing an arbitrary extreme point is picking one data point uniformly at random from the dataset (call it $x^{(i)}$) and, selecting a data point ext such that $argmax_{ext \in X} \|x^{(i)} - ext\|$ is satisfied.

3.1.4 Time Analysis of the Proposed Method

Although there are a few iterations over the distances of the data points with respect to the extreme point, the computational cost of this method is dominated by the complexity of sorting, which is $O(n \log n)$. Both quicksort and heapsort are known to have an average cost of $O(n \log n)$, and can be used as sorting algorithms in our proposed method. Also, bucket sort as an another option could be used, and it has a better average complexity ($O(n)$) than the other two sorting algorithms that mentioned above.

CHAPTER FOUR

Experiments and Results

In this chapter, we compare the results of the proposed seeding technique with two other popular seeding techniques, Forgy's method, and k-means++ initialization. We have used four different classes of datasets to test the performance of each algorithm. The first three kinds are synthetic datasets. One of these synthetic datasets includes clusters that are well-separated in $2, 8, 16, 256, 512$ dimensions which is called CLB-d[1], the second, CL-d[2] contains clusters that are more separated in comparison with the first one, and the third that is called UR-d[3], contains uniform random data in different dimensions. Moreover, there is a fourth class of data, which includes several real-world datasets with different characteristics. The information of the datasets that we have used in our experiments is classified in Table 4.1.

4.1 Clustered Data Generation

The way that clustered datasets are generated, is based on a mathematical definition which is called c-separation (Dasgupta 2000). c-separation is a measurement of how separated the clusters are in a specific dataset. By definition, two Gaussians $N_1(\mu_1, \sigma^2 I_n)$, and $N_2(\mu_2, \sigma^2 I_n)$ are c-separated if

$$||\mu_1 - \mu_2|| \geq c\sigma\sqrt{n} \tag{4.1}$$

[1] **C**Lustered **B**ig dataset. **B**ig represents the size of this dataset which is huge. Also, d in CLB-d represents the dimensionality of the dataset: CLB-2 is a dataset in CLB with dimensionality of 2.

[2] **CL**ustered dataset.

[3] **U**niform **R**andom dataset

[4] (Bache and Lichman 2013)

[5] (Bache and Lichman 2013). Clusters are well-separated.

[6] (Bache and Lichman 2013). Clusters are well-separated.

Table 4.1. Datasets used in experiments

Name	Dimension	# of datapoints	# of clusters	Data description
CL-d	$2, 8, 64, 512$	10000	50	Synthetic clustered
CLB-d	$2, 8, 64, 256, 512$	100000	50	Synthetic clustered
CL-σ	2	10000	10	Synthetic clustered
UR-d	$2, 4, 8, 16, 64, 512$	10000	0	Uniform random
Birch1	2	100000	100	Clusters in regular grid structure
Birch2	2	100000	100	Clusters at a sine curve
Cloud[4]	10	2048	0	Real-world dataset
KDD[5]	74	145751	2000	Biology dataset
Yeast[6]	8	1484	10	Protein localization sites

Based on the above definition, we have calculated the c-separation between each cluster and its nearest cluster in the synthetic clustered datasets in the following way: the distance between each cluster center and its nearest cluster center is calculated, and based on that c is obtained from the Equation 4.1. Therefore, in each dataset there are several c-separations based on the number of clusters. Having several c-separations, for estimating how separated the clusters are, we have calculated the average of them. The average minimum c for CL-d is 7, and for CLB-d is 3[7].

There is another set of data called CL-σ. Figure 4.2 shows these datasets, note that the blue circles are the data points and the yellow dots are the means of the clusters. In these datasets, the only thing that changes is the standard deviation (σ)

[7]Note that based on the definition, A mixture of Gaussians is c-separated if its component Gaussians are pairwise c-separated, but the way that we have calculated the average minimum c-separation is slightly different: instead of calculating the c-separation between every two clusters in a clustered dataset, we have calculated the c-separation between a cluster and its nearest cluster, and then computed the average of these minimum c-separations.

in generating the clusters. The way that we have calculated the c (as in c-separation) in the CL-σ datasets, is described earlier in this section. In the CL-σ datasets, we have 6 different values for c. These values are described in Table 4.3.

4.2 Implementation, Metrics, and the Algorithms Used in the Experiments

We have used three different methods of selecting the initial cluster centers, and then used k-means algorithm to cluster the datasets. In the plots that are illustrated in this chapter, each algorithm's name is abbreviated for simplicity. Table 4.2 shows the complete name of each of the abbreviated names used throughout this chapter (note that k-means++ is used as its original name, and has not been abbreviated).

The algorithms that we have used in the experiments are implemented in C++. Each algorithm consists of two parts: the initialization part, and the k-means algorithm itslef. The algorithms just differ in the initialization part. The k-means part is the same for all of the three methods. The efficient C++ code for k-means algorithm is provided by Dr. Gregory Hamerly.

For the purpose of testing, 20 trials for each case is run and for reporting the results, the average of these values are calculated. The metrics used for comparing the three different techniques, are distortion, overall runtime (seeding runtime + k-means runtime), and the initialization runtime. All the runtimes are measured in seconds. The last metric which is measured, is number of iterations until the algorithms converge. In sections 4.6, and 4.7 for comparing the number of iterations of the algorithms, each algorithm is run 10 times, and the 10 results are illustrated.

Table 4.2. Algorithm's acronym used in the experiments

Acronym	Original name
FG	Forgy's method (chapter 2)
PM	Proposed method (chapter 3)
PMR	Proposed method with using random points as pivot (section 4.9.1)
PMDG1	Proposed method decreased group version 1 (section 4.9.2)
PMDG2	Proposed method decreased group version 2 (section 4.9.2)

4.3 Experimental Results on Synthetic Clustered Data

In this section, the results of runtime and distortion comparison of k-means augmented with three different seeding techniques on CL-d, CLB-d, and CL-σ datasets have been illustrated.

Here we describe the way that we have performed testing on CL-σ datasets. First, a series of tests are done on 6 datasets with 10 clusters such that the Gaussians (the clusters) in the datasets are fixed, and the only thing that differs in these datasets is the distance between clusters. The way that these datasets are generated is first the random centers are generated, then the data points are assigned to each cluster such that the mean of the cluster is one of the random centers and standard deviation or $\sigma = 0.001, 0.001, 0.01, 0.10.5, 1.0$. Therefore the only variable that changes among the datasets is σ. Figures 4.2a, 4.2b, 4.2c, 4.2d, 4.2e, 4.2f illustrate these 6 clustered datasets.

We have calculated c-separation in the CL-σ datasets in the following way: first, the distance between each cluster center and its nearest cluster center is calculated, and based on that, c is obtained from the Equation 4.1. In our case, since there are 10 clusters, in each dataset we get 10 values for c. Therefore, we get the average of these values and assign it to the c as in c-separation. Table 4.3 shows the c-separation value for each CL-σ dataset.

Figure 4.1 illustrates the average runtime of the three algorithms on CL-σ datasets, with $k = 10$. Based on the results, when the clusters are too sparse, or in the case that σ is too small, k-means++ has a better runtime than proposed method + k-means. But, as σ grows, or c-separation decreases, the proposed method + k-means performs better than k-means++. In all cases, FG + k-means runs slower than the other two methods.

During these experiments, we noticed that when $\sigma = 0.0001$, or when the clusters are very well separated, the groups of points generated by the proposed

Table 4.3. CL-σ's c-separation

Sigma	average c-separation
0.0001	6489.18
0.001	708.81
0.01	70.88
0.1	7.08
0.5	1.41
1.0	0.70

method is exactly the same as the number of clusters in the datasets (10), and the distortion is almost the same as the optimal clustering distortion. As σ grows, or the clusters become closer, the number of groups generated by the proposed method increases. Therefore the optimal case for the proposed method is when the clusters in a clustered dataset are too far from each other.

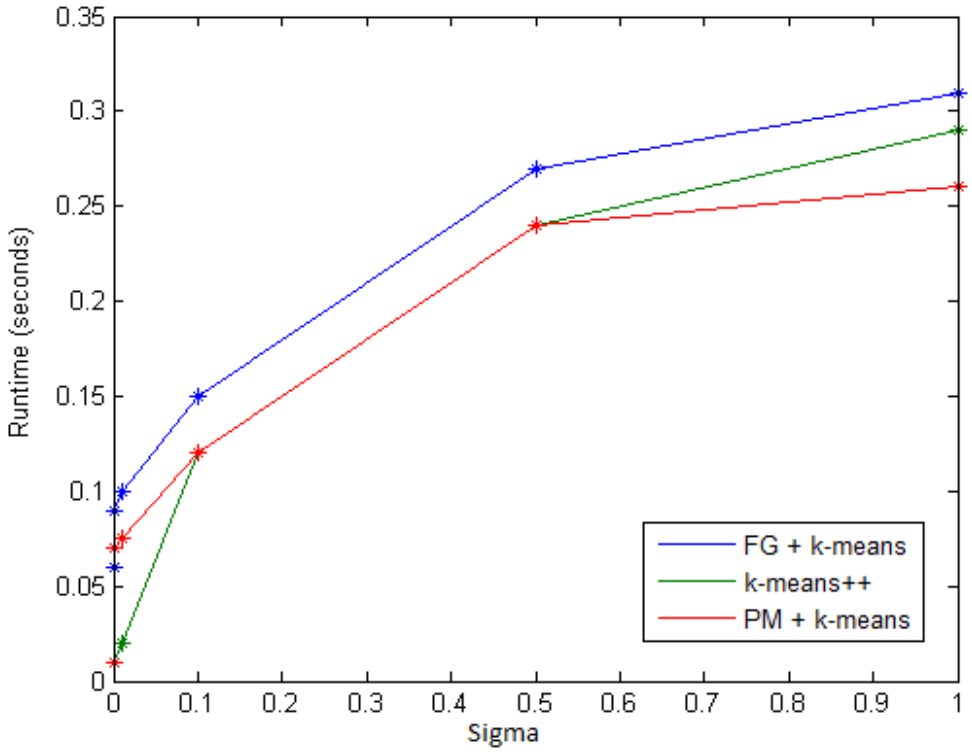
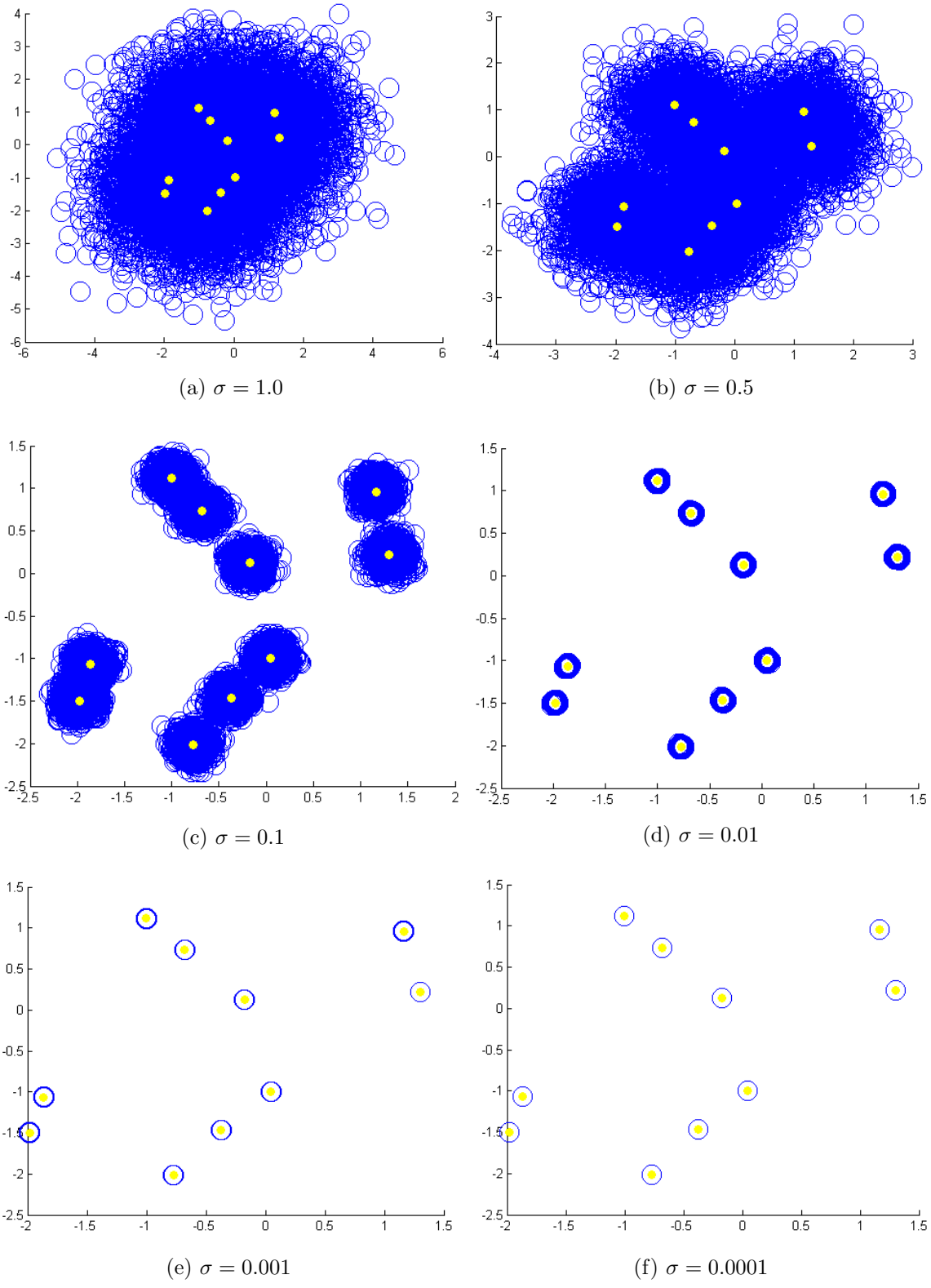


Figure 4.1. Sigma vs. Runtime on CL-σ, with $k = 10$. When Sigma is higher than 0.5, PM + k-means has a better runtime than the other two methods. Higher values of sigma indicate the datasets that look like uniform random datasets, or in other words, clusters in these datasets are very close to each other.



(a) $\sigma = 1.0$

(b) $\sigma = 0.5$

(c) $\sigma = 0.1$

(d) $\sigma = 0.01$

(e) $\sigma = 0.001$

(f) $\sigma = 0.0001$

Figure 4.2. CL-σ datasets

4.3.1 *Initialization Runtime Comparison*

Figure 4.3 shows the k-means++ initialization runtime and the proposed method runtime. Based on the results, k-means++ has a significantly slower runtime for selecting initial cluster centers, specially in high dimensions. But, as the dimension grows the initialization time of the proposed method is growing very slowly. If we consider the asymptotic runtime of the k-means++ initialization which is $O(knd)$, and compare it with the asymptotic runtime of the proposed method which is $O(nd \log nd)$, we see that why k-means++ initialization grows faster than the proposed method especially in higher dimensions.

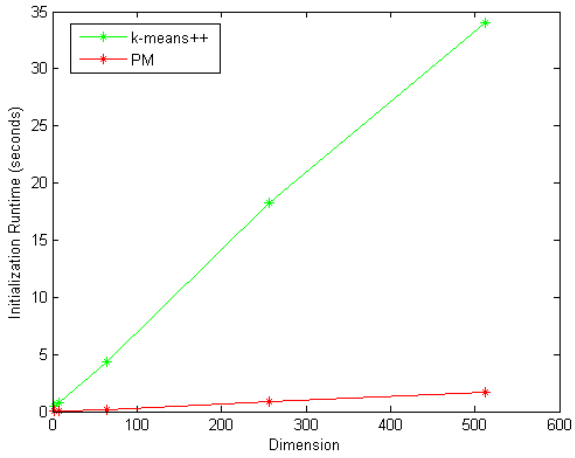


Figure 4.3. Algorithms Initialization Runtime vs. Dimension on CLB-d, with $k = 50$. As dimension grows, PM's runtime increases very slowly while k-means++'s runtime increases quickly. In a very low dimension like 2, the runtimes of both algorithms is very close to each other. In 512 dimension, the runtime of k-means++ is almost 16 times worse than the runtime of PM.

4.3.2 *Groups Generated by the Proposed Method*

Recall that the most important part of the proposed method is generating m groups based on the data points distances to an arbitrary extreme point. Data points that are close to the same distance from the extreme point, would be placed in the same group. Then, based on the number of the groups that are generated we pick k

initial cluster centers. Table 4.4 shows the minimum and maximum number of groups for each synthetic clustered dataset during the 20 trials when $k = 50$.

Table 4.4. Number of groups generated by the proposed method on clustered datasets.

Dataset	min # of groups	max # of groups
CL-2	2105	2555
CL-8	1431	1701
CL-64	469	729
CL-512	115	188
CLB-2	3524	3681
CLB-8	15983	18065
CLB-64	7782	8770
CLB-256	3981	5798
CLB-512	2040	3912

The number of the groups generated by the proposed method depends on the distribution of the data points in the space, or in other words, it depends on the distribution of the distance of the data points with respect to an extreme point. For example, assume we have two d-dimensional datasets. Now, consider we perform the proposed method on these datasets with an extreme point in each of them. In the first dataset the distance distribution of the data points with respect to an extreme is illustrated in Figure 4.4.

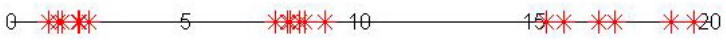


Figure 4.4. The distribution of the distance of the data points with respect to an extreme point. Based on the distribution, we see there are three groups of the distances. If we perform the proposed method on these distances, we notice that 3 groups are generated.

If the distribution of the distances of the data points with respect to the extreme point is like the above, the proposed method generates 3 groups, because the average of the difference between each two consecutive distances is less than the

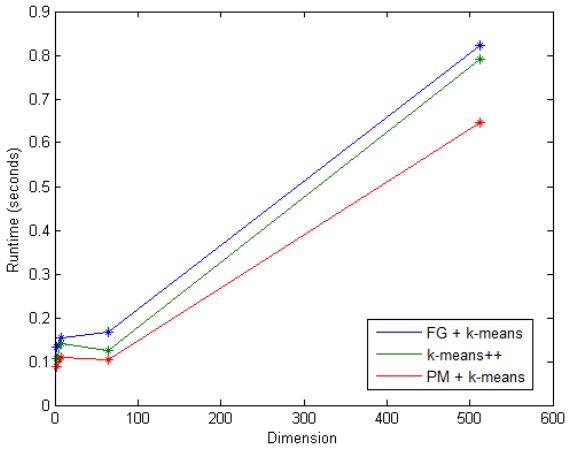
difference between each two consecutive distances in most of the cases (except two case). Therefore, we would have a small number of groups. Now consider a different distribution of the distances of the data points with respect to an extreme point in the second dataset which is illustrated in Figure 4.5.

Figure 4.5. The distribution of the distance of the data points with respect to an extreme point. Based on the distribution, it is not clear at the first glance that how many groups are generated by the proposed method. But if we perform the proposed method on these distances, the number of groups that are generated is 7.

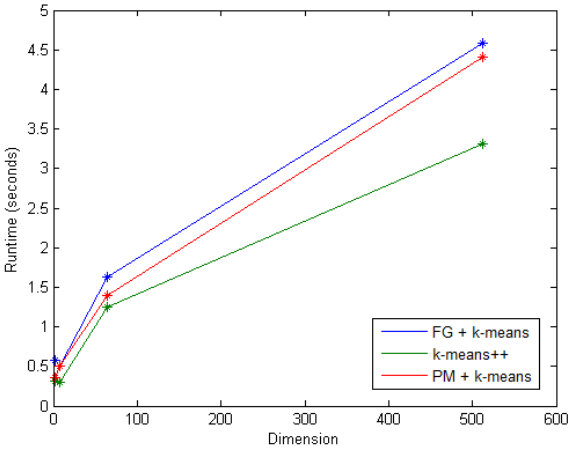
In Figure 4.5, the number of groups generated by the proposed method would be greater than the previous case. Back to the Table 4.4, and based on the above facts, the reason that number of groups generated by the proposed method is much larger than the number of clusters in a clustered dataset, is because of the the distribution of the distances of the data points with respect to the selected extreme point. In our previous experiments, we noticed that if the clusters are very well separated, the number of the groups generated by the proposed method is the same as the number of clusters in the dataset, while in the CL-d and CLB$-d$ datasets, since the clusters are just separated (not very well), the number of the groups generated by the proposed method is much larger than the number of clusters in these datasets. Therefore, we may conclude that the number of the groups generated by the proposed method is heavily dependant on the distribution of the data points, and the extreme point that is selected.

4.3.3 Runtime Comparison

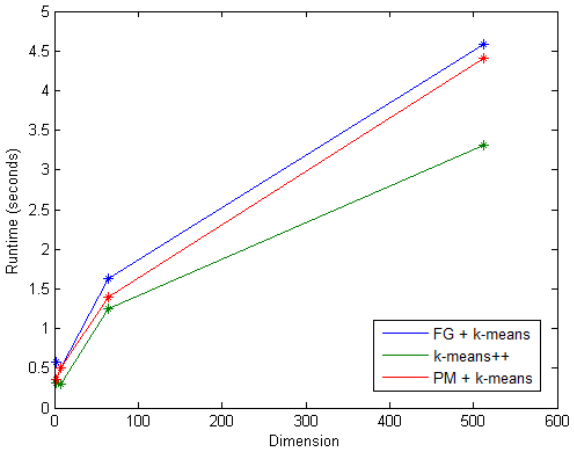
Figures 4.6a, 4.6b, and 4.6c illustrate the results of comparing the average runtime of k-means augmented with three different initialization techniques: Forgy's initialization method, k-means++ initialization, and the proposed method.



(a) Runtime vs. Dimension on CL-d, $k = 10$



(b) Runtime vs. Dimension on CL-d, $k = 50$



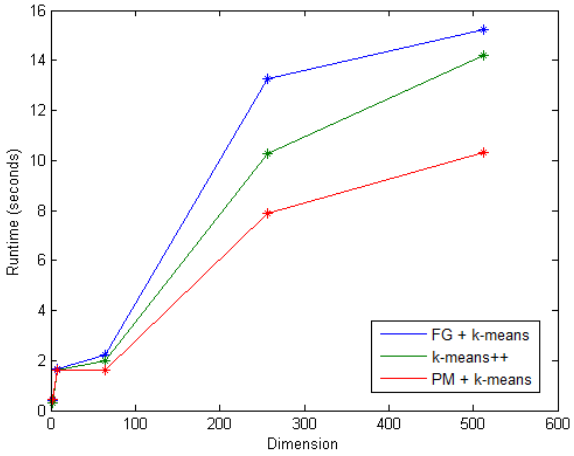
(c) Runtime vs. Dimension on CL-d, $k = 75$

Figure 4.6. (a) Runtime vs. Dimension on CL-d with $k = 10$. PM + k-means has the best runtime. (b) Runtime vs. Dimension on CL-d with $k = 50$. k-means++ has the best runtime. When dimension is 512, gap between k-means++ and the other two methods increases. (c) Runtime vs. Dimension on CL-d with $k = 75$. The Runtime of all the methods is very close to each other especially in dimensions less than 100.

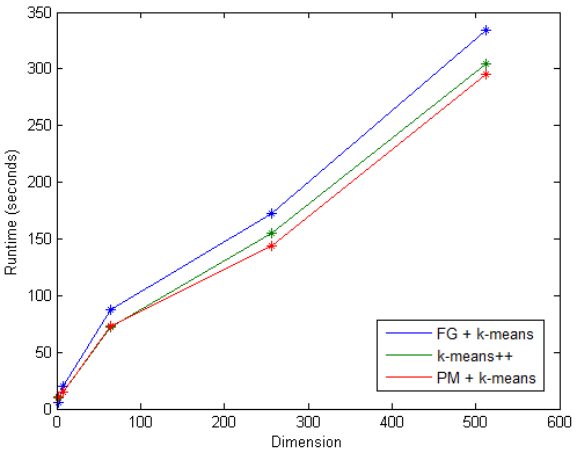
These tests are performed on CL-d datasets. Based on the results, the proposed method and k-means++ always have better runtimes than FG + k-means. When it comes to comparison of the proposed method and k-means++, results show that when k is less than or greater than the number of clusters ($k = 10$ or $k = 75$), the proposed method even has a better runtime than k-means++. Also, based on

the results in low dimensions, all three versions of the k-means have close runtimes especially when $k = 50, 75$.

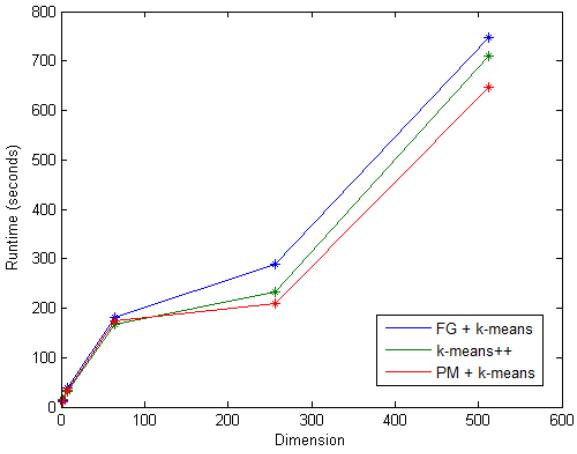
Figures 4.7a, 4.7b, and 4.7c illustrate the results of comparing the average runtime of k-means augmented with three initialization techniques on CLB-d datasets.



(a) Runtime vs. Dimension on CLB-d, $k = 10$



(b) Runtime vs. Dimension on CLB-d, $k = 50$

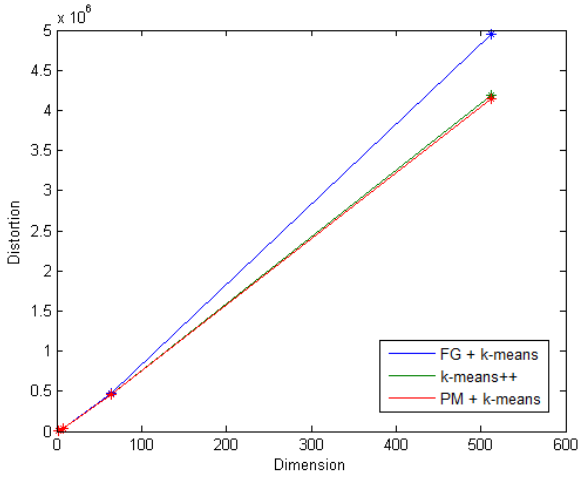


(c) Runtime vs. Dimension on CLB-d, $k = 75$

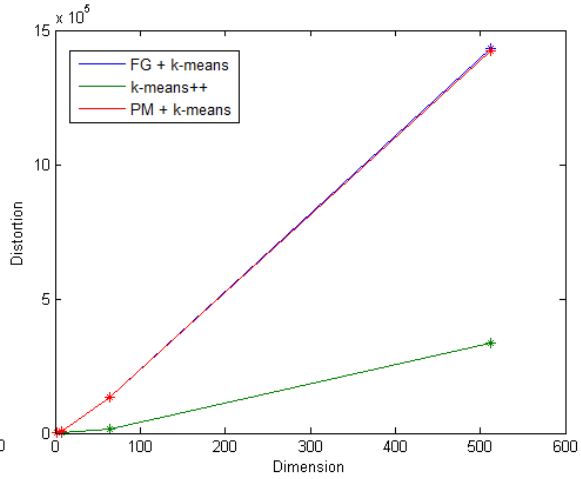
Figure 4.7. (a) Runtime vs. Dimension on CLB-d with $k = 10$. k-means augmented with PM has the best runtime. (b) Runtime vs. Dimension on CLB-d with $k = 50$. The runtime of PM + k-means and k-means++ are very close to each other. (c) Runtime vs. Dimension on CLB-d with $k = 75$. In low dimension (less than 100), all three method have almost the same runtime.

4.3.4 *Distortion Comparison*

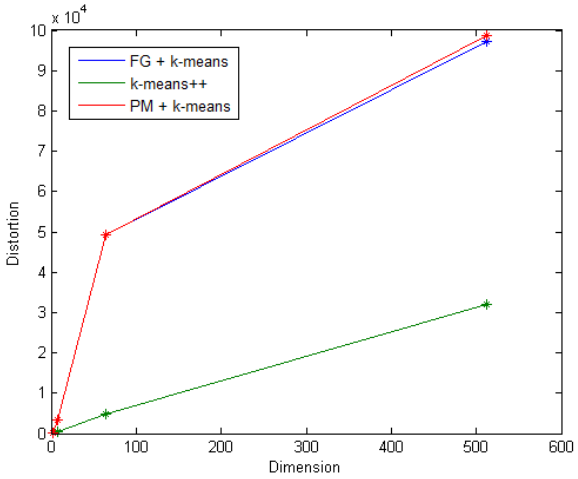
Figures 4.8a, 4.8b, and 4.8c illustrate the results of comparing the final average distortion of k-means augmented with three different initialization techniques; Forgy's, initialization method of k-means++, and proposed method.



(a) Distortion vs. Dimension on CL-d, $k = 10$



(b) Distortion vs. Dimension on CL-d, $k = 50$

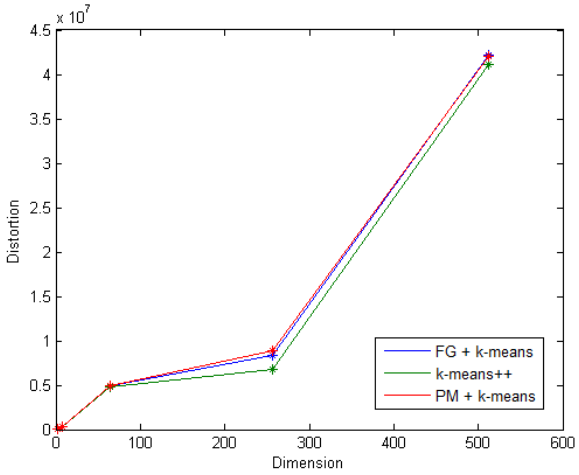


(c) Distortion vs. Dimension on CL-d, $k = 75$

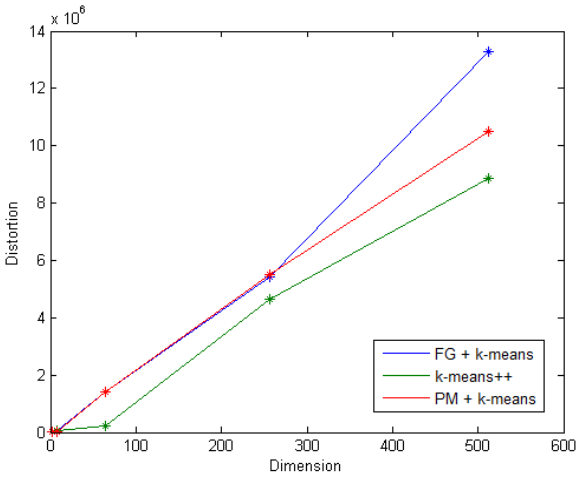
Figure 4.8. (a) Distortion vs. Dimension on CL-d with $k = 10$. FG + k-means has the worst distortion, while the other two methods have very close distortion. (b) Distortion vs. Dimension on CL-d with $k = 50$. k-means++ has a significantly better distortion than the other two methods. while the other two methods have a very similar distortion. (c) Distortion vs. Dimension on CL-d with $k = 75$. FG + k-means and PM + k-means have very similar distortion, and worse than k-means++.

These tests are performed on CL-d datasets. Based on the results, k-means++ has a significantly better average distortion value at the time of convergence in comparison with k-means augmented with Forgy's method, and k-means augmented with the proposed method. But there is a reason for that which comes from the nature of the k-means++. The reason is that k-means++ unlike the other two methods, is proved to be $O(\log n)$-competitive with the optimal clustering, which is a great property of k-means++. As we mentioned in chapter two, although this does not tell that it will reach the global optimum, it does tell that it will get reasonably close to the global optimum which is of great importance. On the other hand, it seems that when k is small and less than the number of clusters in a clustered dataset, k-means augmented with the proposed method is performing as well as k-means++, and these two are performing slightly better than k-means augmented with Forgy's method.

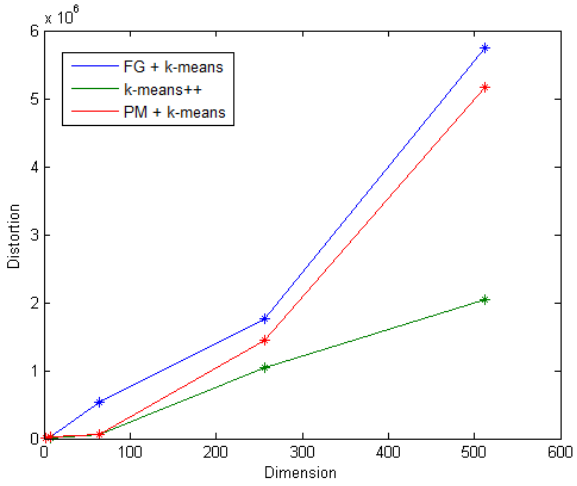
Figures 4.9a, 4.9b, and 4.9c illustrate the results of comparing the final average distortion on CLB-d datasets. Based on the results, k-means++ in all the three cases wins the other two methods. In the case that $k = 10$, in low dimensions, all three methods have a very close average distortion, but as dimension grows, the difference between k-means++ average distortion, and the average distortion of the other two methods increases, also FG + k-means, and PM + k-means have similar average distortions in this case. When $k = 50$, k-means++ has a much better average distortion in all the dimensions. In this case, FG + k-means, and PM + k-means have a very close average distortion up to 256, but when dimension is 512, PM + k-means has a much better distortion than FG + k-means. When $k = 75$, still the average distortion of k-means++ is better than the other two methods especially in higher dimensions. Also, PM + k-means has a better average distortion than FG + k-means almost in all dimensions. Again, the reason that k-means++ has a better average final distortion compared with the other two methods, is the same that we mentioned in the above paragraph.



(a) Distortion vs. Dimension on CLB-d, $k = 10$



(b) Distortion vs. Dimension on CLB-d, $k = 50$



(c) Distortion vs. Dimension on CLB-d, $k = 75$

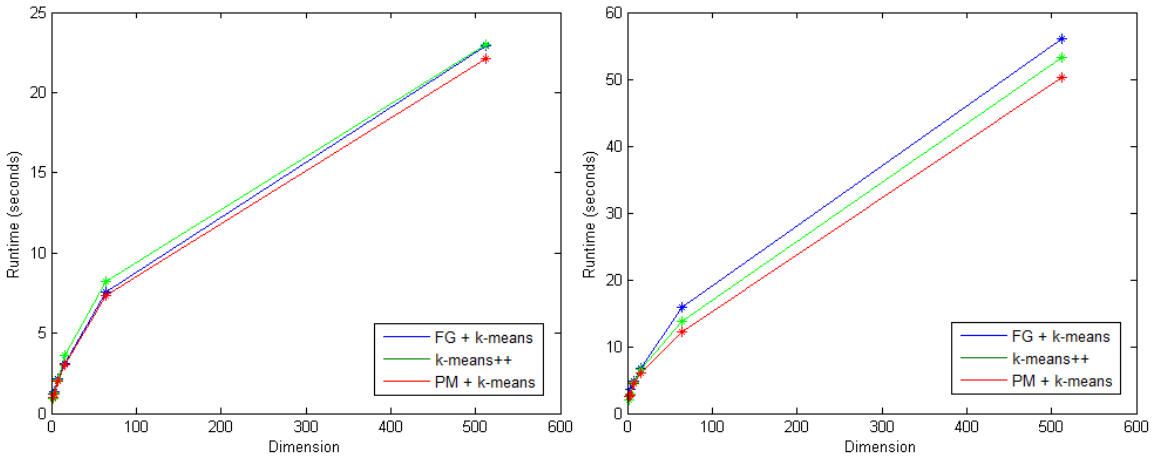
Figure 4.9. (a) Distortion vs. Dimension on CLB-d with $k = 10$. All three methods have a very close distortion especially up to dimension 128. In higher dimensions, k-means++ has a slightly better average distortion than the other two methods, but FG + k-means and PM + k-means have a very similar average distortion in all the dimensions. (b) Distortion vs. Dimension on CLB-d with $k = 50$. FG + k-means and PM + k-means have a very close distortion up to 256 dimension, while k-means++ has a much better distortion than these two on average. (c) Distortion vs. Dimension on CLB-d with $k = 75$. k-means++ has the best distortion in comparison with the other two methods, and when it gets to the higher dimensions, the gap between the k-means++ distortion and the other two methods increases significantly. Also, PM + k-means has a better average distortion than FG + k-means.

4.4 Experimental Results on Uniform Random Data

In this section, the results of comparison of three algorithms with respect to their average runtime and distortion on UR-d datasets is illustrated. Two different values for k have been used in the experiments on UR-d datasets: 50 and 200. The way that we have generated the UR-d datasets is using a Matlab function which returns 10000 pseudorandom scalar drawn from the standard uniform distribution on the open interval $(0, 1)$.

4.4.1 Runtime Comparison

Figures 4.10a, and 4.10b show the results of comparing the average runtime of k-means augmented with three different seeding techniques in seconds.



(a) Runtime vs. Dimension on UR-d, $k = 50$ (b) Runtime vs. Dimension on UR-d, $k = 200$

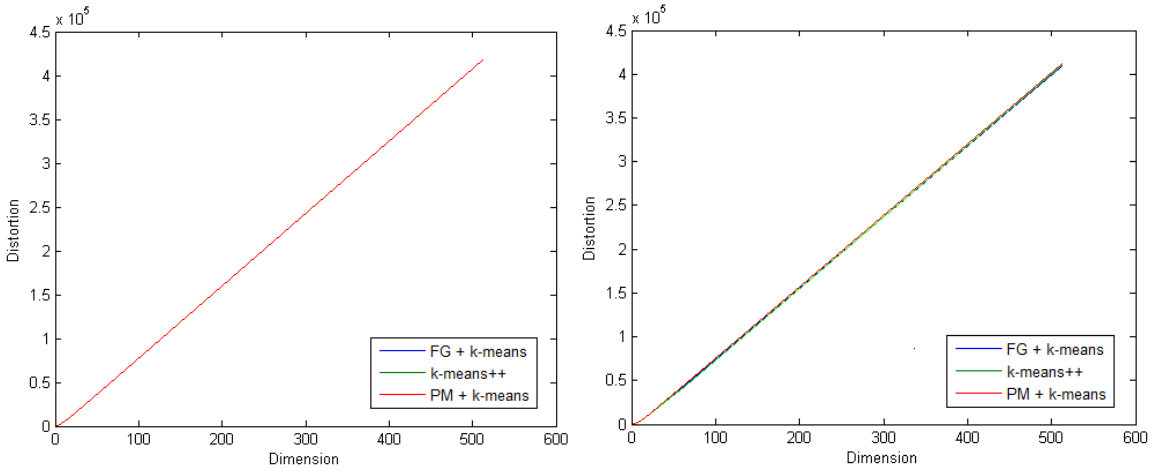
Figure 4.10. (a) Runtime vs. Dimension on UR-d with $k = 50$. All the three methods have similar runtimes especially in lower dimesnions. (b) Runtime vs. Dimension on UR-d with $k = 200$. k-means++ have a slightly better runtime than the other two methods. In low dimensions, all the three methods have a very close runtime.

Based on the results, when $k = 50$, all three algorithms have almost the same average runtime. Specifically, in low dimensions which are below 100, FG + k-means and k-means augmented with the proposed method have slightly a better runtime

than k-means++, while in high dimensions, k-means augmented with the proposed method performs slightly better than the other two methods. In the case that $k = 200$, k-means++ has a better runtime than the other two methods in both low and high dimensions. When it comes to the comparison between proposed seeding + k-means and FG + k-means, proposed method + k-means has a better runtime than FG + k-means. But, in general, proposed method + k-means runtime is closer to k-means++ than to FG + k-means. Moreover, in low dimensions $(2, 4, 8, 16)$, in both cases, all three methods have very similar average runtime.

4.4.2 Distortion Comparison

Figures 4.11a, and 4.11b show the results of comparing the average final distortion of k-means augmented with three different seeding techniques.



(a) Distortion vs. Dimension on UR-d, $k = 50$ (b) Distortion vs. Dimension on UR-d, $k = 200$

Figure 4.11. (a) Distortion vs. Dimension on UR-d with $k = 50$. All three methods have the same distortion in all the dimensions. (b) Distortion vs. Dimension on UR-d with $k = 200$. Like Figure 4.11a, all the three methods have a very close distortion on different dimensions. The reason for this behavior is described above.

When $k = 50$, all three algorithms have a very close average distortion, both in low and high dimensions. Also, when $k = 200$, the average distortion is almost the

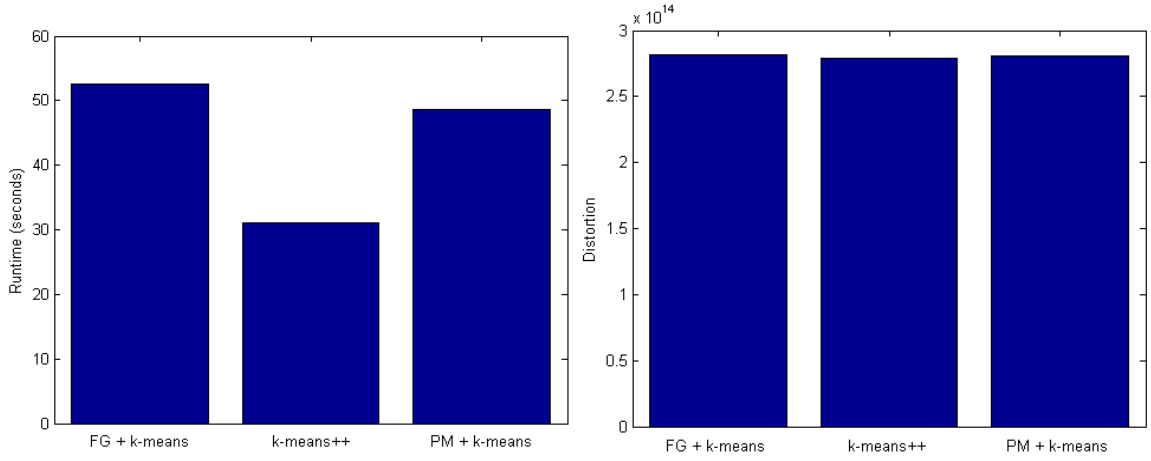
same for all three methods. This raises the question: Why FG + k-means has the similar average distortion when as with the other two methods that are considered to choose initial cluster centers more accurately? The reason is behind the fact that uniform random data does not have a specific structure and data points are not well-clustered. Consider a 2-dimensional uniform random dataset with 1000 data points that we want to perform k-means on, with $k = 4$. Since the data is distributed randomly, we can consider 2×2 grid that divides the 2-dimensional space into 4 regions. Now, we want to select 4 samples as our initial cluster centers from the dataset. The optimal clustering suggests that we choose the 4 samples from each of the 4 subspaces. Next, assume that the same dataset is evenly distributed among four well-separated clusters in 2-dimensional space. In this case, the optimal clustering suggests that the initial cluster center be chosen from each of the 4 clusters. For the uniform random dataset, since Forgy's method chooses centers uniformly at random from the dataset, consider a worst case which all 4 samples are chosen from the same subspace, and for the clustered dataset, as the worst case scenario, consider all 4 centers are chosen from the same cluster. In the case of uniform random data, the number of data points assigned to each center are close, and this causes that k-means converges to a local optimum that is close to the optimal solution even if it takes more iterations. While in the case of clustered data, the number of data points assigned to each cluster is not close to each other at all; a huge portion of data points may be assigned to some centers, just a few data points are assigned to the other centers, and this makes when k-means converges, the solution be far away from the optimal solution. Therefore, in the case of uniform random data, the FG + k-means method would have similar results as the other two methods, even if the other two methods choose initial cluster center more accurately. In other words, no matter what we use as a seeding technique, running k-means on uniform random data almost always converges to a local optimum very close to global optimum.

4.5 Experimental Results on the Real-World Datasets

In this section we illustrate the results of comparison of three algorithms with respect to their average runtime and distortion on 4 real-world datasets. The information about these datasets are described in Table 4.1. We have measured the average metrics with $k = 50$.

4.5.1 Runtime and Distortion Comparison on the Birch1 Dataset

Figures 4.12a, and 4.12b show the results of comparing the average runtime, and distortion of k-means augmented with three different seeding techniques on the Birch1 dataset.



(a) Average runtime on Birch1, $k = 50$

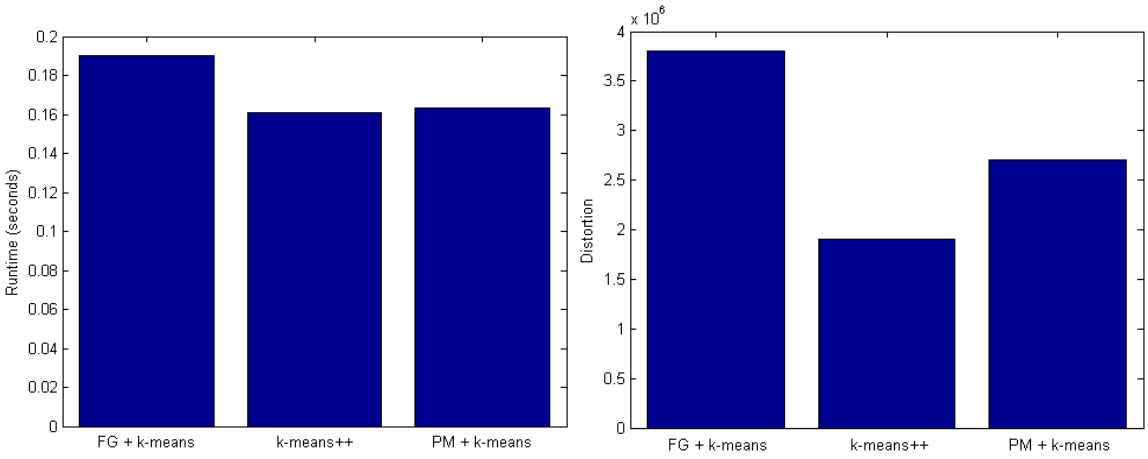
(b) Average Distortion on Birch1, $k = 50$

Figure 4.12. (a) Average runtime on Birch1 with $k = 50$. k-means++ has a better runtime than the other two methods, because it converges with fewer iterations. (b) Average Distortion on Birch1 with $k = 50$. Average distortion is almost the same for all the three methods.

Based on the results, the average runtime of k-means++ on the Birch1 dataset is much better than the average runtime of the k-means augmented with the other two methods. Also, the average runtime of k-means augmented with the proposed method is slightly better than the average runtime of k-means augmented with Forgy's method. When it comes to the distortion, all three methods have similar values.

4.5.2 *Runtime and Distortion Comparison on the Cloud Dataset*

Figures 4.13a, and 4.13b show the results of comparing the average runtime, and distortion of k-means augmented with three different seeding techniques on the Cloud dataset.



(a) Average runtime on Cloud, $k = 50$

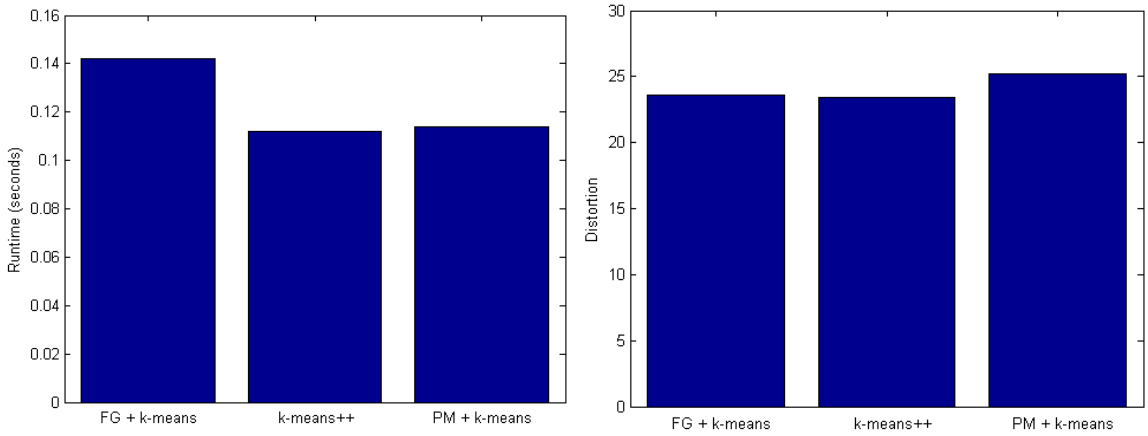
(b) Average Distortion on Cloud, $k = 50$

Figure 4.13. (a) Average runtime on Cloud with $k = 50$. FG + k-means is slightly slower than the other two method, because it converges with higher number of iterations. (b) Average Distortion on Cloud with $k = 50$. k-means++ has the best distortion in comparison with the other two methods. Also, the proposed method wins the FG + k-means.

Based on the results of the above plots k-means++, and k-means augmented with the proposed method have similar average runtimes that are better than the average runtime of k-means augmented with the Forgy's method. Also, when it comes to the distortion comparison, k-means++ has a significantly better distortion than the other two methods, while k-means augmented with Forgy's method is having a really poor distortion in comparison with k-means++. On the other hand, the average distortion of k-means augmented with the proposed method is in the middle of the other two methods: it is better than k-means + Forgy's, and it is worse than k-means++.

4.5.3 *Runtime and Distortion Comparison on the Yeast Dataset*

Figures 4.14a, and 4.14b show the results of comparing the average runtime, and distortion of k-means augmented with three different seeding techniques on the Yeast dataset.



(a) Average runtime on Yeast, $k = 50$

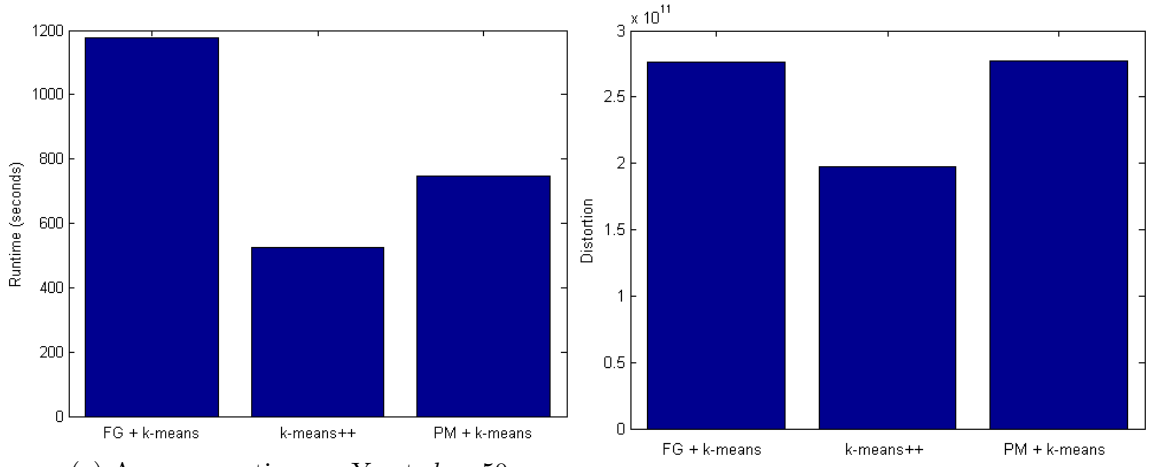
(b) Average Distortion on Yeast, $k = 50$

Figure 4.14. (a) Average runtime on Yeast with $k = 50$. k-means++ and the PM + k-means have close runtimes, and better than FG + k-means. (b) Average Distortion on Yeast with $k = 50$. All three methods have a close distortion.

Based on the results, the average runtime of k-means++ and k-means augmented with the proposed method are almost the same, and better than the average runtime of k-means augmented with Forgy's method. On the other hand, the average distortion of k-means++ and k-means augmented with Forgy's method is slightly better than than the average distortion of k-means augmented with the proposed method.

4.5.4 *Runtime and Distortion Comparison on the KDD Dataset*

Figures 4.15a, and 4.15b show the results of comparing the average runtime, and distortion of k-means augmented with three different seeding techniques on the KDD dataset.



(a) Average runtime on Yeast, $k = 50$

(b) Average Distortion on Yeast, $k = 50$

Figure 4.15. (a) Average runtime on KDD with $k = 50$. k-means++ has the best runtime, and FG + k-means has the worst performance. (b) Average Distortion on KDD with $k = 50$. k-means++ has the best distortion, but the other two methods have a very close distortion.

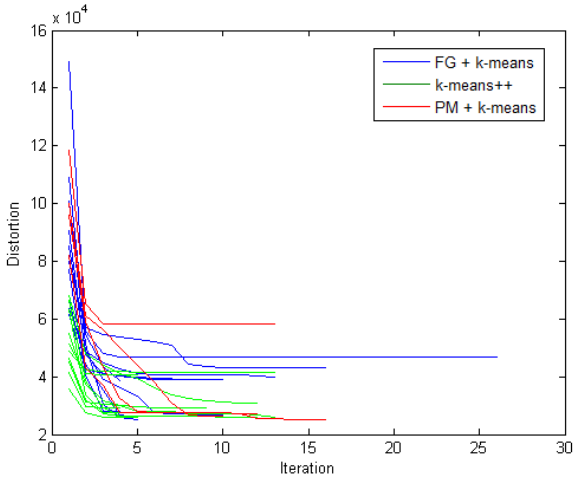
Based on the results, k-means augmented with Forgy's method is running much slower than the other two methods on average. k-means++ has the best average runtime, and k-means augmented with the proposed method is almost in between. When it comes to the distortion comparison, the average distortion of k-means++ is better than the other two methods, while k-means augmented with the proposed method, and k-means augmented with Forgy's method have a very close average distortion.

4.6 Number of Iterations vs. Distortion Comparison on CLB-2, 256

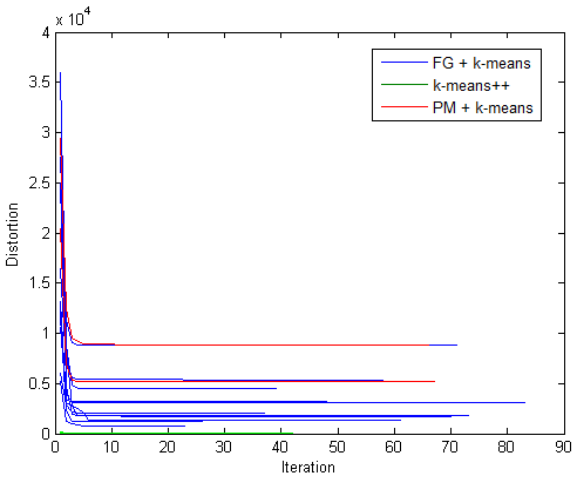
In this section, each algorithm is run 10 times, and the distortion at the end of each iteration is reported until k-means converges. In each dataset that is used in this section, three different values for k are used: 10, 50, 75.

Figures 4.16a, 4.16b, and 4.17c illustrate the results of number of iterations vs. distortion on CL-2 with three different value for k: 10, 50, and 75. Based on the results, except the case that $k = 10$, proposed method causes k-means converge

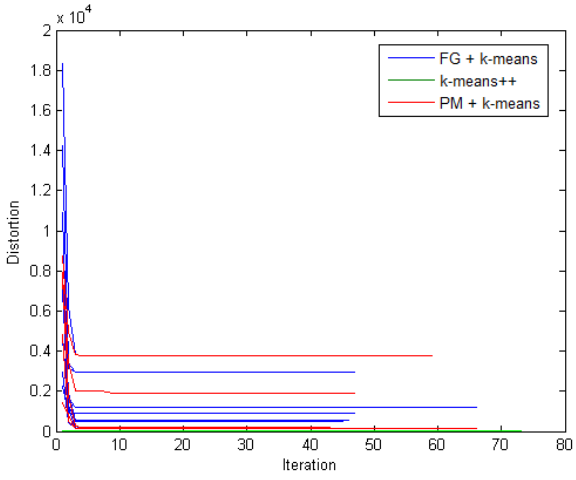
with fewer iterations. The interesting thing is that when $k = 75$, k-means++ could be even worse than FG + k-means in some cases. Note that for k-means++, we see a single green line, but actually it is 10 lines that have a slope very close to each other. These lines may have different iteration, but the worst iteration of these line is greater than 70, and the best of them is 56.



(a) Iteration vs. Distortion on CLB-2, $k = 10$



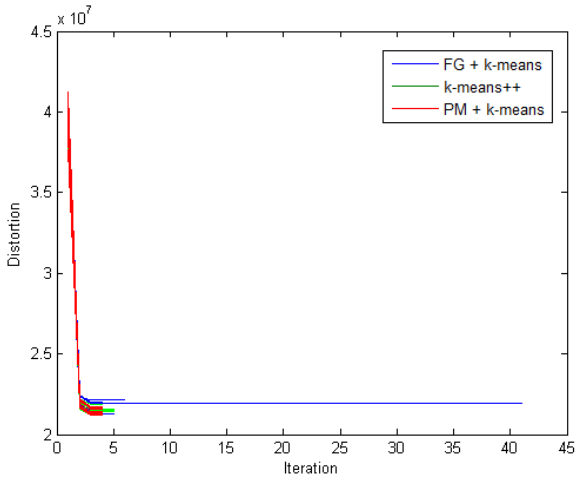
(b) Iteration vs. Distortion on CLB-2, $k = 50$



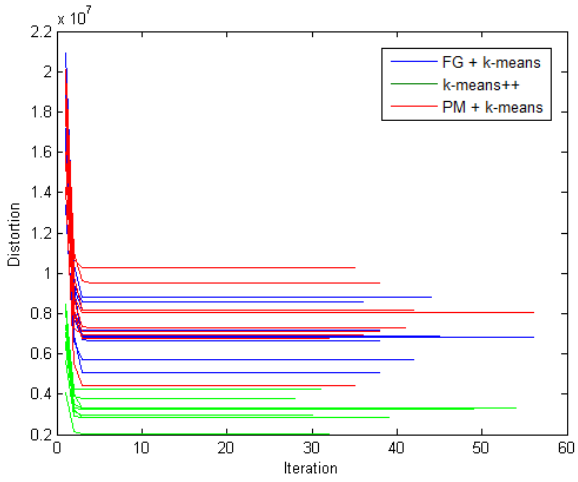
(c) Iteration vs. Distortion on CLB-2, $k = 75$

Figure 4.16. (a) Iteration vs. Distortion on CLB-2, $k = 10$. PM + k-means and k-means++ are performing better than FG + k-means. (b) Iteration vs. Distortion on CLB-2, $k = 50$. k-means++ starts with a very low distortion, and converges faster than the other two methods. (c) Iteration vs. Distortion on CLB-2, $k = 75$. k-means++ starts with a very small distortion, but has the worst number of iterations.

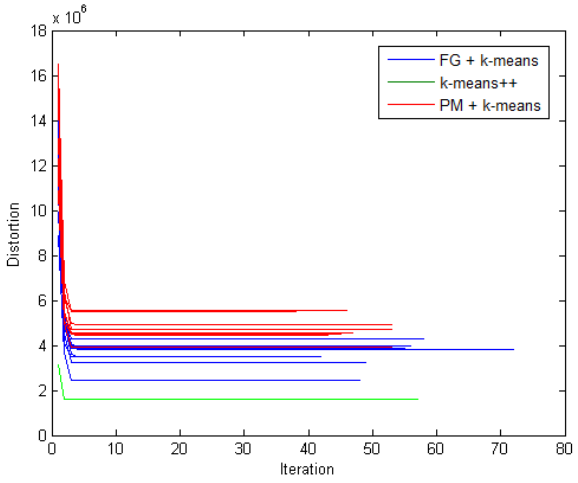
Figures 4.17a, 4.17b and, 4.17c illustrate the results of number of iterations vs. distortion of k-means augmented with Forgy's method, k-means++ initialization, and proposed method on CL-256 dataset. Like the previous case we have used three different values for k: 10, 50, and 75.



(a) Iteration vs. Distortion on CLB-256, $k = 10$



(b) Iteration vs. Distortion on CLB-256, $k = 50$



(c) Iteration vs. Distortion on CLB-256, $k = 75$

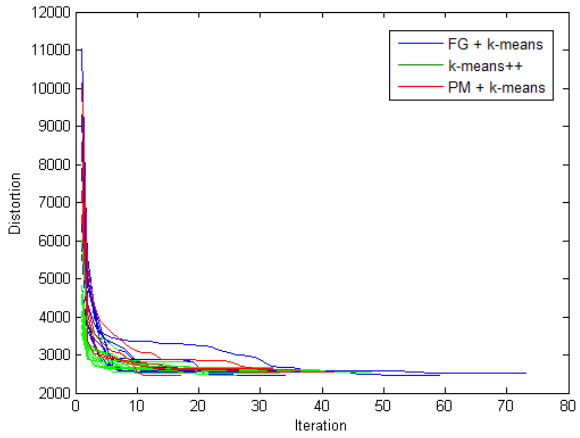
Figure 4.17. (a) Iteration vs. Distortion on CLB-256 with $k = 10$. PM + k-means converges faster than the other two methods FG + k-means is converging much slower on average in this case. Also, k-means++ is converging slightly slower than PM + k-means on average. (b) Iteration vs. Distortion on CLB-256 with $k = 50$. k-means++ converges with a better distortion, but the number of iterations is not much better than the other two methods. (c) Iteration vs. Distortion on CLB-256 with $k = 75$. FG + k-means converges slower than the other two methods.

Based on the results when $k = 10$, the proposed method + k-means has fewer iterations than the other two methods, and FG + k-means is doing very poor in comparison with the other two methods. On the other hand, when $k = 50$, there is not a huge gap between the methods, but still FG + k-means is the worst of all in terms of the number of iterations, but still k-means++ has the smallest number of iterations among all. When $k = 75$, proposed method + k-means converges with fewer number of iterations on average. FG + k-means has the maximum number of iterations in our 10 experiments. When it comes to initial distortion, k-means++ starts with a much better distortion than the other two methods.

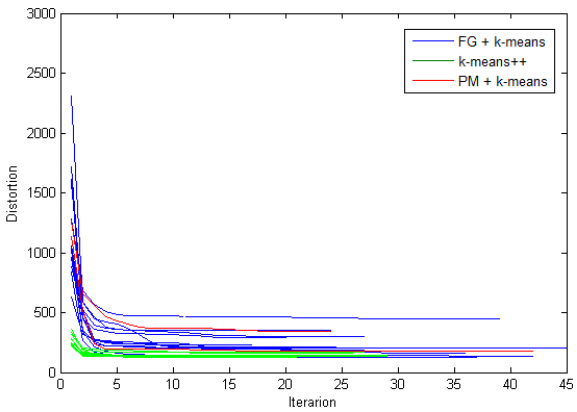
4.7 Number of Iterations vs. Distortion Comparison on CL-2, 64

In this section like the previous section, each algorithm (k-means++, k-means augmented with Forgy's method, and k-means augmented with the proposed method) has been run 10 times, and the distortion at the end of each iteration is reported. In each dataset that is used in this section, three different values for k are used: 10, 50, and 75.

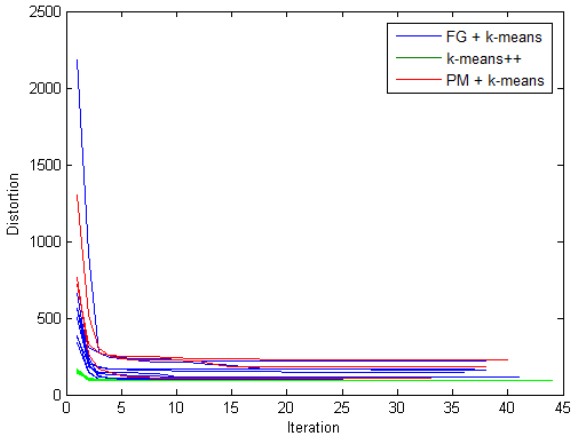
Figures 4.18a, 4.18b, and 4.18c show the the result of number of iterations vs. distortion on CL-2 with $k = 10, 50, 75$. Based on the results when $k = 10$, FG + k-means has more iterations than the other two methods, and it also starts with a greater distortion value. Also, k-means++ starts with the lowest distortion and proposed method + k-means's initial distortion is between these two. When the algorithms converge, the distortion seems to be the same for all of them. When $k = 50$, the maximum number of iterations and greatest initial distortion belong to FG + k-means. k-means++, starts with the lowest distortion value and converges with fewer iterations than the other two methods. proposed method is in the between of these two methods. When $k = 75$, k-means++ starts at a very low distortion which is good, but it takes almost as much iterations as the other two methods. Also, FG + k-means and PM + k-means start at relatively great distortions.



(a) Iteration vs. Distortion on CL-2, $k = 10$



(b) Iteration vs. Distortion on CL-2, $k = 50$

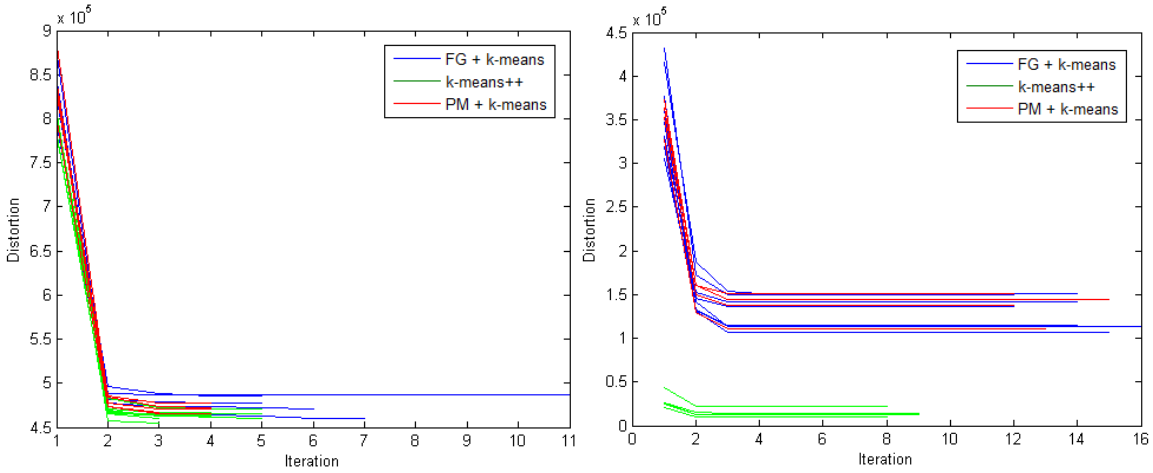


(c) Iteration vs. Distortion on CL-2, $k = 75$

Figure 4.18. (a) Iteration vs. Distortion on CL-2 with $k = 10$. FG + k-means converges much slower than the other two methods. (b) Iteration vs. Distortion on CL-2 with $k = 50$. k-means++ converges faster than the other two methods. (c) Iteration vs. Distortion on CL-2 with $k = 75$. k-means++ may converge with higher number of iterations ate least in some cases, but it has the best distortion.

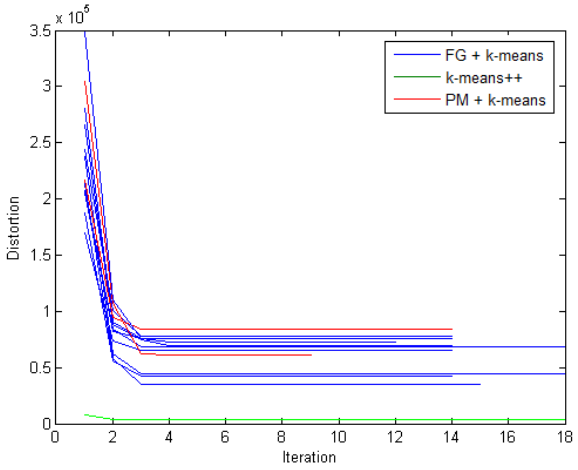
Figures 4.19a, 4.19b and, 4.19c illustrate the results of number of iterations vs. distortion on CL-64 with three different value for k: 10, 50, and 75. When $k = 10$, k-means augmented with Forgy's method converges with higher number of iterations in comparison with the other two methods. k-means augmented with the proposed method is converging with fewer number of iterations in comparison with the other two methods on average. When $k = 50$, k-means++ is doing better than

the other two methods both in terms of the distortion, and the number of iterations. When $k = 75$, k-means++ is converging with more number of iterations on average in comparison with k-means augmented with the other two seeding techniques. Also, k-means augmented with the proposed method is converging faster on average.



(a) Iteration vs. Distortion on CL-64, $k = 10$

(b) Iteration vs. Distortion on CL-64, $k = 50$



(c) Iteration vs. Distortion on CL-64, $k = 75$

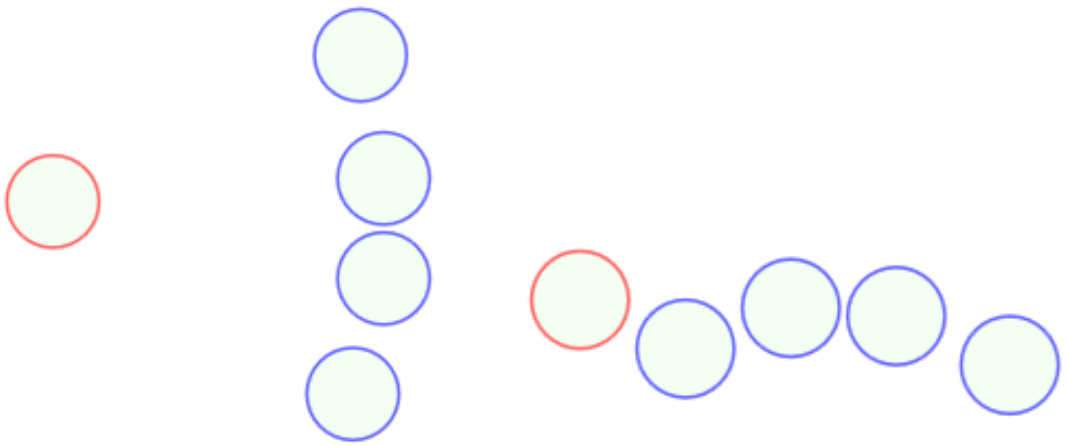
Figure 4.19. (a) Iteration vs. Distortion on CL-64 with $k = 10$. FG + k-means converges with higher number of iterations in most of the cases. (b) Iteration vs. Distortion on CL-64 with $k = 50$. k-means++ has the best performance both in terms of distortion and number of iterations. (c) Iteration vs. Distortion on CL-64 with $k = 75$. k-means++ has the best distortion, but it may converge with higher iterations at least in some cases.

4.8 Further Discussion

In this section, we discuss the main drawbacks of the proposed method, and try to explain the cases that the algorithm behaves poorly in terms of the number of iterations and distortion. Also, we discuss the case that the proposed method finds a good set of initial cluster centers for the k-means algorithm such that it leads k-means converge with a few number of iterations.

4.8.1 Dataset Structure and the Proposed Method

Consider Figures 4.20a, and 4.20b which represent two different datasets in 2-dimensional space with some clusters. Each circle in these figures represents a cluster with a fixed number of data points. Note that the blue clusters in Figure 4.20a have been placed on the circumference of an imaginary circle with origin equal to the mean of the data points in the red cluster, and radius r.



(a) (b)

Figure 4.20. Two different clustered datasets

Assume we want to find a set of initial cluster centers to seed the k-means with. Also, assume in both datasets, the extreme point is chosen from the red clusters. Based on the proposed method, we need to calculate the distance of the remaining

data points with respect to the extreme point. Therefore, in Figure 4.20a, all the data points would have a distance equal to $r \pm \epsilon$ with respect to the extreme point ($\epsilon << r$). Now consider the dataset in Figure 4.20b. In this case, the distance of the data points with respect to the extreme point can not be estimated, because data points are placed in different locations with respect to the extreme point, and we have a wide range of distances, in contrast to the dataset in Figure 4.20a. In other words, there is more variation in the distances of the data points in Figure 4.20b than in Figure 4.20a. Based on these facts, we expect the proposed method works poorly on the datasets which look like the dataset in Figure 4.20a, because the number of groups generated by the proposed method would be really small (even fewer than the number of clusters in the dataset) in that case, and this causes data points that belong to the different clusters, be placed in the same group generated by the proposed method, and it leads k-means converge to a poor solution with many iterations. On the other hand, if the datasets be more similar to the dataset in Figure 4.20b, the results in terms of number of iterations would be much better, and the probability that the data points in the different clusters be placed in the same groups generated by the proposed method would be much less than the first case.

Based on this intuition, we have conducted a test on two synthetic clustered datasets: Birch1 and Birch2 (see Table 4.1 for details about these datasets). Birch2 (Figure 4.21b) looks more like the dataset in Figure 4.20b, and it is composed of some clusters that form a sine curve, while Birch1 which is composed of some clusters in regular grid structure, looks more like the dataset in Figure 4.20a. We have tested k-means augmented with the proposed method on these datasets 20 times, and the results are illustrated in Table 4.5. Based on the results in Table 4.5, proposed method selects a better set of initial cluster centers in Birch2 dataset than in Birch1 in terms of the number of iterations. Actually, the proposed method works perfectly on Birch2, because it only requires 3 iterations for k-means to converge.

Table 4.5. Clustering results on Birch1 and Birch2 datasets with $k = 100$.

Dataset	Maximum # of iterations
Birch1	385
Birch2	3

Based on the facts and experiments in this section, we can further improve the result of the algorithm on a dataset by choosing some extreme point(s) that make k-means algorithm converge with fewer number of iterations.

(a) Birch1 dataset

(b) Birch2 dataset

Figure 4.21. Birch Datasets

4.8.2 Good Extreme Point Candidates

In this part, with the assumptions that we made and test in the previous subsection, and with the help of some mathematical definition, we may be able to get better results (in terms of number of iterations) when we use the proposed method as a seeding technique.

Based on the definition, variance measures how far a set of numbers is spread out. In other words, if variance is zero it means that all the values are identical. In Figure 4.20a, since the distance of the data points with respect to the extreme point

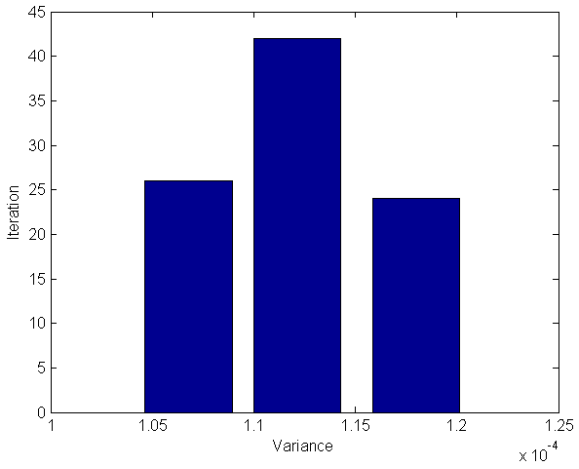
is $r \pm \epsilon$, and the values are very close to each other, therefore the variance would be small. On the other hand, for the dataset illustrated in Figure 4.20b, the variance would be high. If we define the sample variance be an unbiased estimator for the population variance, it would be(Forbes, Evans, Hastings, and Peacock 2011, Kenney and Keeping 1954):

$$s_{n-1}^2 \equiv \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2 \tag{4.2}$$

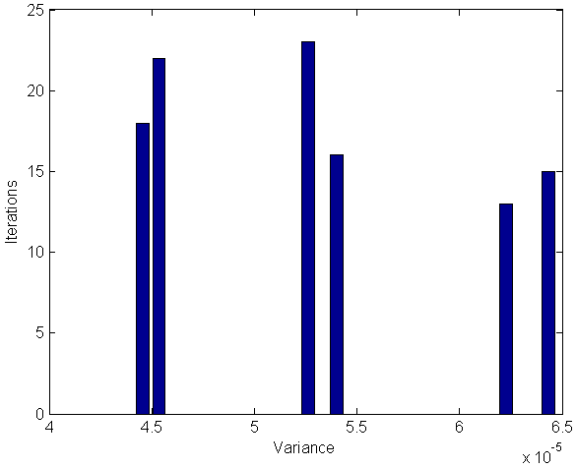
In general, if we perform the proposed method on a specific dataset, with several extreme points, and compute the variance of the distance of the remaining data points with respect to the extreme point, the extreme points that cause a sorting that results in distances with high variance on average, lead to selecting better initial cluster centers, and these good initial centers cause k-means converge with fewer iterations.

Therefore, we have a priori knowledge about the final results of k-means algorithm. Since the proposed seeding technique is a fast method, we can run the proposed method several times, with different extreme points, and start k-means with the sets of initial clusters centers that have higher variance on average. Figures 4.22a, 4.22b, 4.22c, 4.23a, and 4.23b show the results of clustering on CL-d and CLB-d datasets. For each dataset, the proposed method is run 20 times, and the unique results in terms of the number of iterations of the k-means algorithm, and the variance of the distances are reported. Note that the variance that is reported in the figures in this subsection, is the variance of the distance of the data points with respect to the extreme point in each run.

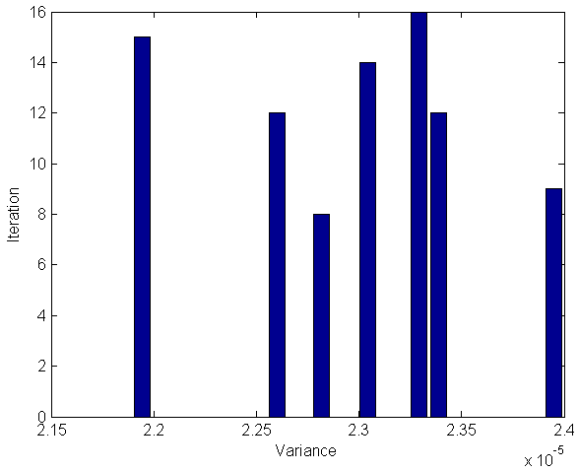
Based on the results, the extreme points that generated higher variance on average are better candidates, and we can use them to choose the initial cluster centers. Therefore, we can avoid having too many iterations by running the proposed method several times, and calculate the average variance, and then avoid using the



(a) Variance vs. Iteration on CL-2, $k = 50$



(b) Variance vs. Iteration on CL-8, $k = 50$

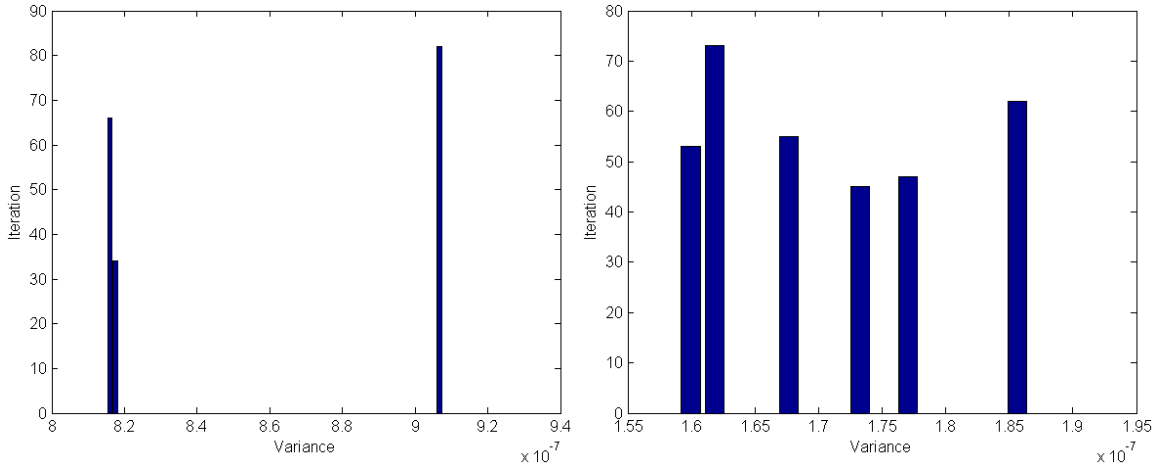


(c) Variance vs. Iteration on CL-64, $k = 50$

Figure 4.22. (a) Variance vs. Iteration on CL-2 dataset with $k = 50$. (b) Variance vs. Iteration on CL-8 dataset with $k = 50$. (c) Variance vs. Iteration on CL-64 dataset with $k = 50$.

extreme points that cause variances lower than the average variance. In this case, we are able to improve the results of the k-means algorithm. Although, this may not be true all the time. For example, in Figure 4.23a, on the CLB-2 dataset, the assumptions that we have made is not valid anymore, and the extreme point that generated the highest variance, has the worst number of iterations, while the other two extreme points that generated lower variances, cause k-means converge with fewer

iterations. In general the experiments in section 4.8.2 do not guarantee that always we can get fewer iterations with choosing extreme points that cause higher variances, but our intuition is that in the datasets that are similar to Figure 4.21b, are better candidates to use the proposed method as a seeding technique.



(a) Variance vs. Iteration on CLB-2, $k = 50$ (b) Variance vs. Iteration on CLB-64, $k = 50$

Figure 4.23. (a) Variance vs. Iteration on CLB-2 dataset with $k = 50$. In contrast to our assumptions, the extreme point that generated the highest variance, has the worst number of iterations. (b) Variance vs. Iteration on CLB-64 dataset with $k = 50$. In this case, the extreme point with the highest variance has more number of iterations on average. Also, the extreme points with highest variances, have fewer number of iterations in comparison with the other extreme points. In general, extreme points that have lower variances on average, converge with higher number of iterations, and this follows what we assumes previously.

Based on the discussions in this section, we conclude that one of the main drawbacks of this method is its dependency on the dataset structure. One of the other drawbacks of the proposed method is that in most of the cases the number of groups generated by the proposed method is much greater than the number of clusters in the dataset (Table 4.4), and these groups are likely to be noisy. In sections 4.9.2, and 4.9.3 we try to see if we can improve the results by reducing the number of groups in the dataset.

4.9 Lesion Studies

In this section, we try to test some alternatives to the proposed method in order to see whether we can improve it in terms of the number of iterations and/or distortion, in other words we want to see if we slightly modify the proposed seeding technique (Algorithm 4), is it possible to make k-means converge to a better distortion and fewer iterations, or not. For this reason, first we try a different way of selecting a pivot point in the first step of the proposed method (recall that based on Algorithm 4, we use extreme points as pivot points). The reason for trying a different pivot point other than an extreme point, is that it may help to put the data points in the same clusters into the same groups generated by the proposed method. In another lesion study, we aim at decreasing the number of groups generated by the proposed method in two different ways. The reason for trying to reduce the number of groups is that in most of the cases (see 4.4) the number of groups are much more than the number of clusters in the clustered datasets that we tested earlier in this chapter, and based on this, we know that there may be data points that belong to the same groups but not actually to the same clusters. Therefore, decreasing the number of groups may help to put the data points from the same clusters to the same groups with higher probability.

4.9.1 An Alternative to Extreme Point

Based on Algorithm 4, we group the data points based on their distances to an extreme point in the dataset. Throughout reading this thesis, a question might be raised whether there are other data points in the dataset except the extreme points that may lead the k-means algorithm converge with fewer iterations and possibly a better distortion. Therefore, we have slightly changed the Algorithm 4 such that instead of choosing extreme points at the first step of this algorithm, we choose uniformly random chosen data points from the dataset, and we call this modified version PMR algorithm which is described in Algorithm 5. As you see Algorithm 5

steps, the only difference with this algorithm and Algorithm 4 is the first step of these two algorithms.

Algorithm 5 PMR - Proposed method with using random points

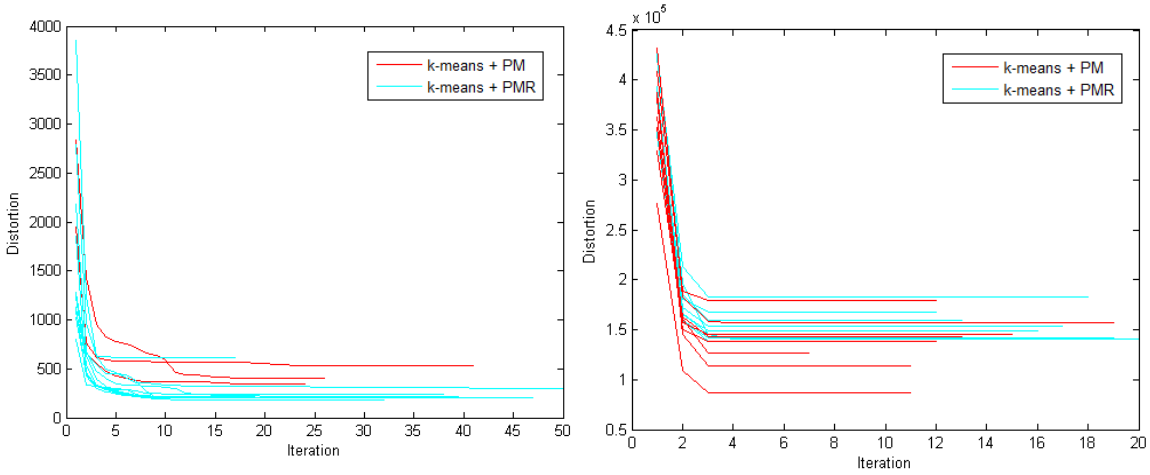
1: **procedure** PMR-SEEDING(X,K)
2: $r \leftarrow$ choose a data point uniformly a random form X
3: **for** $i \in N$ **do**
4: $dist(i) \leftarrow \|x^{(i)} - r\|$
5: The rest is the same as the original Algorithm 4

For testing how well the PMR algorithm chooses the initial cluster centers, and compare it with the original proposed method, we have conducted a test on the CL-2, 64 datasets and illustrated the results in Figures 4.24a, and 4.24b. The algorithms we have used in these experiments, are proposed method with using extreme points for grouping the data points (Algorithm 4), and the modified version of the proposed method with using the uniformly random chosen data points from the dataset (Algorithm 5). We have run k-means 10 times for each algorithm while we have used a fixed $k = 50$ in each run of the k-means algorithm. Based on the results, in general the proposed method with using extreme points works slightly better than the other version, because it makes k-means converge with a better distortion and fewer iterations. Also, it seems that on the CL-64 dataset, the proposed method with using extreme points, makes k-means converge to a significantly better distortion in some cases.

4.9.2 Decreasing the Number of Groups

In this part, we have used two different methods for decreasing the number of groups generated by the proposed method. One of the reasons for trying a modified version of the proposed method which generates fewer groups is that we have the chance that the number of groups be less than k, therefore we could try to pick more than one point from some of the groups (Algorithm 4 line 14), and this is a case that

in our previous experiments has not happened, because in our previous experiments almost always the number of groups generated by the proposed method was greater than k.



(a) Iteration vs. Distortion on CL-2, $k = 50$

(b) Iteration vs. Distortion on CL-64, $k = 50$

Figure 4.24. (a) Iteration vs. Distortion on CL-2 with $k = 50$. In most of the cases, the number of iterations caused by k-means + PMR is greater than k-means + PM. (b) Iteration vs. Distortion on CL-64 with $k = 50$. In some cases, k-means + PM converges to a significantly better distortion in comparison with k-means + PMR.

Recall that in Algorithm 4, lines 6, and 7, the difference between each two consecutive distances is calculated, and then we compute the average of these differences as a threshold for knowing which data points should be grouped together. Here we want to use a slightly different way for calculating the mean and generating the groups: instead of calculating the difference of each two consecutive distances, we need to skip $p \in \mathbb{N}$ distances at a time and calculate the difference between a distance and the distance which is placed after skipping p distances, in other words, if the distances are w, x, y, z, and $p = 1$ instead of calculating $w - x, x - y$, and $y - z$, we calculate $w - y$, and $x - z$ (in this example we have skipped 1 distance at a time). This modified version of the proposed method is described in Algorithm 6. One more important thing is the way the we generate the groups at line 9 of Algorithm 6 which

is different from the original version of the proposed method. We describe this step with a simple example: assume that the sorted distances are $a, b, c, d, e, f, g, h, i$, and $p = 2$. Now for generating the groups, we start from a, and calculate the differences like: $a - d, b - e, c - f, d - g, e - h, f - i$. Now, we split the distances only when $a - d > avg$, and $b - e > avg$, and $c - f > avg$, or $d - g > avg$, and $e - h > avg$, and $f - i > avg$.

One thing to notice in Algorithm 6 is that when we want to find the average value at line 7, we calculate the difference between a distance and the distance which is placed after skipping p distances, and this causes that the differences be larger, and therefore, when we want to calculate the average of these differences, average would be greater than in the case that we calculate the average of each two consecutive distances as in Algorithm 4. Based on this, in general, we would get fewer groups when we use Algorithm 6 for grouping the data points.

Algorithm 6 PMDG1 - Proposed method with decreased groups version 1

```
1:  procedure PMDG1-SEEDING(X,K,P)
2:      ext ← choose an arbitrary extreme point from X
3:      for i ∈ N do
4:          dist(i) ← ||x^(i) − ext||
5:      sort(dist)                          ▷ Sort the distances from step 4
6:      for i ∈ N − (p + 1) do
7:          diff(i) ← |dist(i) - dist(i + p + 1) |
8:      avg ← average(diff)                 ▷ Calculate the average of values in step 7
9:      G ← partition(dist,avg)             ▷ Divide the distance list into m groups
10:     The rest is the same as the original Algorithm 4
```

For testing how well this modified version selects the initial seeds, we have augmented k-means with Algorithm 6, and Algorithm 4, and run k-means 10 times with these two seeding techniques. Figures 4.25a, 4.25b, 4.25c and 4.25d show the results of comparing the proposed method and its alternative we discussed here. In Figures 4.25a, and 4.25b, the value for p in the alternative algorithm we discussed

above is 10, and 20 respectively, and for the Figures 4.25c, and 4.25d, the value for p is 2, and 4 respectively. Based on the results, proposed method is doing a better job in selecting the initial cluster centers, because it makes k-means converge to a better distortion and in fewer iterations on average. Also, Table 4.6 shows the number of groups generated by the PMDG1 algorithm.

Table 4.6. Number of groups generated by the PMDG1 algorithm on CL-2, 64.

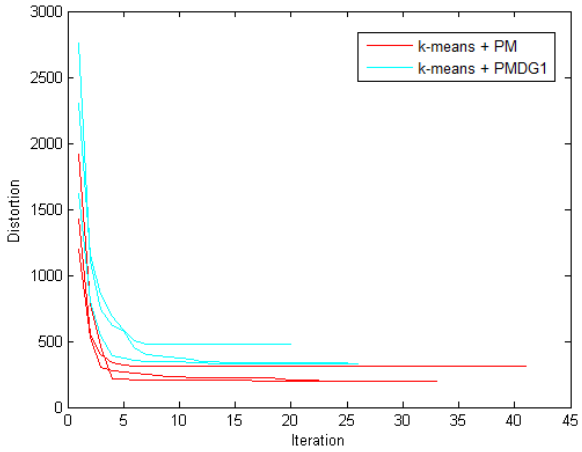
Dataset	max # of groups	min # of groups
CL-2 with $p = 10$	144	96
CL-2 with $p = 20$	91	63
CL-64 with $p = 2$	58	34
CL-64 with $p = 4$	37	23

Based on the results in Tables 4.4, and 4.6, the average number of groups generated by the original version of the proposed method is much greater than the average number of groups generated by the PMDG1 algorithm.

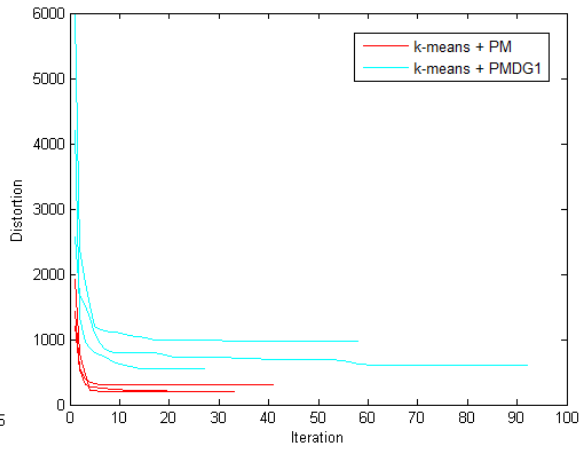
The next method for decreasing the number of groups is increasing the average of the difference of the consecutive distances (Algorithm 4, line 8). This modified version of Algorithm 4 is called PMDG2 and the steps of this algorithm is described in Algorithm 7.

Algorithm 7 PMDG2 - Proposed method with decreased groups version 2

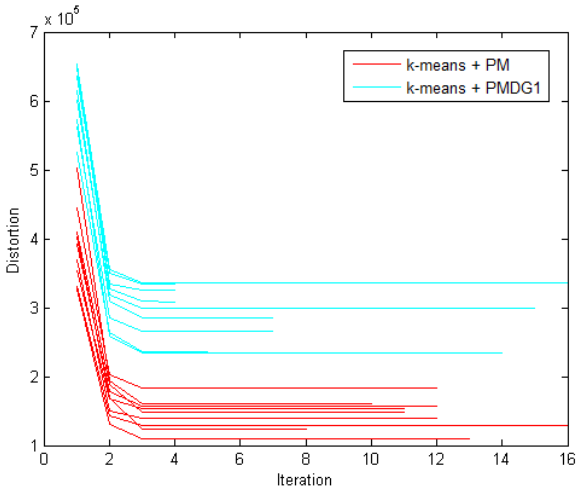
1: **procedure** PMDG2-SEEDING(X,K,T)
2: *ext* ← choose an arbitrary extreme point from X
3: **for** $i \in N$ **do**
4: $dist(i) \leftarrow \|x^{(i)} - ext\|$
5: *sort(dist)* ▷ Sort the distances from step 4
6: **for** $i \in N - 1$ **do**
7: *diff(i)* ← |*dist(i)* - *dist(i+1)* |
8: avg ← average(*diff*) ▷ Calculate the average of values in step 7
9: threshold ← avg × T
10: $G \leftarrow$ partition(*dist,threshold*) ▷ Divide the distance list into m groups
11: The rest is the same as the original Algorithm 4



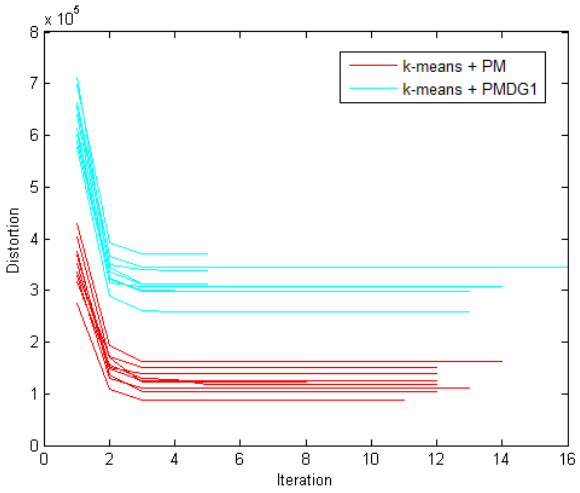
(a) Iteration vs. Distortion on CL-2, $p = 10$



(b) Iteration vs. Distortion on CL-2, $p = 20$



(c) Iteration vs. Distortion on CL-64, $p = 2$



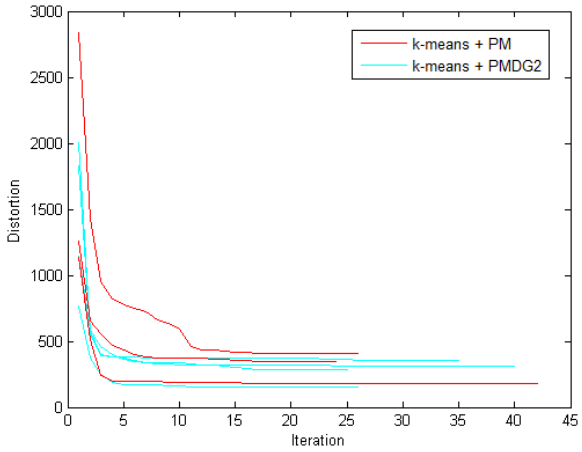
(d) Iteration vs. Distortion on CL-64, $p = 4$

Figure 4.25. (a) Iteration vs. Distortion on CL-2 with $k = 50$, and $p = 10$. Based on the results, the average final distortion of k-means + PMDG1 is better than the average final distortion of k-means + PM, but the average initial distortion of k-means + PM is better than the other method. Also, when it comes to the number of iterations, k-means + PMDG1 converges faster on average than k-means + PM. (b) Iteration vs. Distortion on CL-2 with $k = 50$, and $p = 20$. k-means + PM converges much faster in comparison with k-means + PMDG1. Also when it comes to the initial and final distortion, k-means + PM is doing much better on average than k-means + PMDG1. (c) Iteration vs. Distortion on CL-64 with $k = 50$, and $p = 2$. The initial and final distortion of k-means + PM is significantly better than the initial and final distortion of k-means + PMDG1. When it comes to the number of iterations, in some cases k-means + PMDG1 is converging much faster than k-means + PM. (d) Iteration vs. Distortion on CL-64 with $k = 50$, and $p = 4$. Both initial and final distortion of k-means + PM is much better than the initial and final distortion of k-means + PMDG1. But k-means + PMDG1 converges much faster than k-means + PM in some cases.

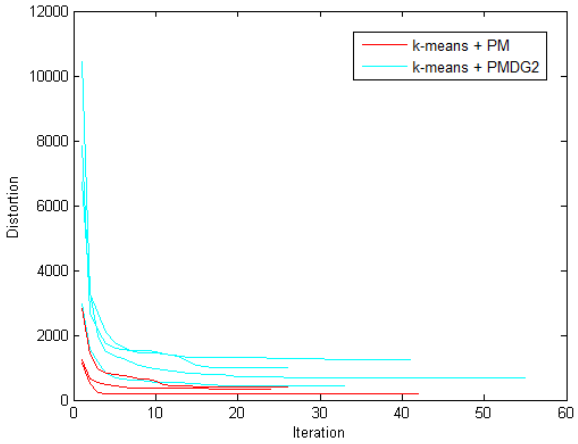
We have conducted a test on CL-2, 64 datasets, and run k-means augmented with Algorithm 4, and Algorithm 7. We have used two different values of average for each dataset; for CL-2 dataset we have used average $\times 2$, average $\times 10$, and for CL-64 dataset we have used average $\times 2$, and average $\times 5$. Figures 4.26a, 4.26b, 4.26c, and 4.26d illustrate the results of comparing the proposed method with the modified version of proposed method that we just discussed. Based on the results, in all the cases the original version of proposed method converges to a better distortion, but when it comes to comparing the number of iterations, both of the algorithms have almost similar results on average. Also, Table 4.7 shows the number of groups generated by the modified version of the proposed discussed above. By increasing the average value, the number of groups decreases significantly in some cases. The reason for using average $\times 2$, and average $\times 5$ when testing the algorithms on CL-64 dataset is that if we use greater values for the average, we will get very few groups, and in some cases we have seen that even the number of groups can be as few as just 3, 4, or 5. In general we are not interested in generating a few groups or too many groups, because if either of these two cases happen, the groups are more noisy. By noisy, we mean that the data points that belong to the same cluster have been placed in different groups. For example, if we have 50 clusters in our dataset, and we have generated 5 groups, it means that data points that belong to the different clusters have been put together into the same groups, or if we have 2500 groups while the dataset contains 50 clusters, it means that there are so many data points that belong to the same clusters, but have been placed in different groups.

Table 4.7. Number of groups generated by the PMDG2 algorithm on CL-2, 64.

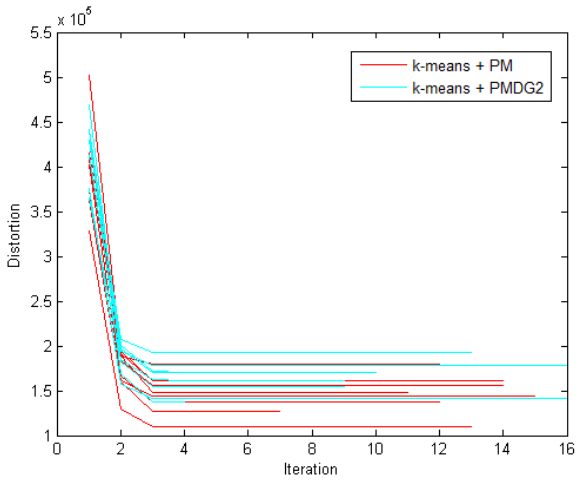
Dataset	max # of groups	min # of groups
CL-2 with average $\times$ 2	1452	1121
CL-2 with average $\times$ 10	982	812
CL-64 with average $\times$ 2	201	133
CL-64 with average $\times$ 5	59	28



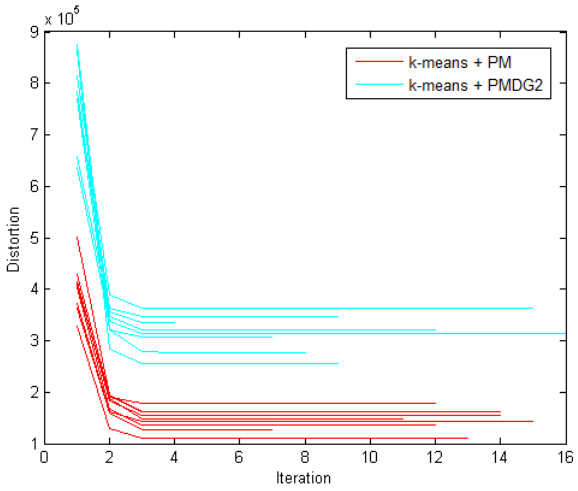
(a) Iteration vs. Distortion on CL-2, $ave = 2$



(b) Iteration vs. Distortion on CL-2, $ave = 10$



(c) Iteration vs. Distortion on CL-64, $ave = 2$



(d) Iteration vs. Distortion on CL-64, $ave = 5$

Figure 4.26. (a) Iteration vs. Distortion on CL-2 with $k = 50$, and average $\times 2$. Based on the results, The average final distortion of both cases does not differ that much. Also when it comes to the average number of iterations, again both algorithm are close to each other. If we consider the initial distortion, based on the results the average initial distortion of k-means + PMDG2 is slightly better than the average initial distortion of k-means + PM. (b) Iteration vs. Distortion on CL-2 with $k = 50$, and average $\times 10$. k-means + PM converges to a better distortion. Also, the initial distortion of k-means + PMDG2 is much worse than the initial distortion of the k-means + PM. (c) Iteration vs. Distortion on CL-64 with $k = 50$, and average $\times 2$. Both algorithms have almost similar pattern of convergence. (d) Iteration vs. Distortion on CL-64 with $k = 50$, and average $\times 5$. The initial distortion of k-means + PM is much better than the initial distortion of k-means + PMDG2, and this causes k-means + PM converge to a better distortion.

CHAPTER FIVE

Conclusion

Clustering is the unsupervised classification of observations, data points, or feature vectors into clusters (groups). The clustering problem has been addressed in many contexts and by researchers in many disciplines; this reflects its broad appeal and usefulness as one of the steps in exploratory data analysis. The k-means algorithm is one of the most popular clustering techniques because of its speed and simplicity. This algorithm is very simple and easy to understand and implement. The first step of this algorithm is choosing k initial cluster centers. The way the this set of initial cluster centers are chosen, have a great effect on speed and quality of k-means.

Two of the most popular seeding techniques are Forgy's method and k-means++ initialization. Forgy's method is efficient in practice because it choose the initial cluster centers uniformly at random from the dataset. The main draw back of this method is that the final solution of k-means using this method is far from the optimal clustering, and usually this causes k-means convergence require many iterations. On the other hand, k-means++ initialization is considered a slow seeding technique because it needs k passes over the data set for choosing the initial cluster centers, but this method has much better results in terms of speed and quality in comparison with Forgy's method.

In this thesis we introduced a new and fast seeding technique, that is doing a better job (in most of the cases) in terms of runtime than k-means++ and k-means augmented with Forgy's seeding technique. But when it comes to the quality of clustering k-means++ is doing a great job on almost all the datasets. More specifically, in large clustered datasets with high dimensionality ($n > 10000$, $d > 64$), k-means augmented with proposed method converges faster than the other two methods. Moreover, the proposed method works fine on clustered datasets such that the

clusters are not too far or clusters that are 2-separated or 3-separated on average. When it comes to uniform random datasets, in high dimensions, k-means augmented with proposed method also generates better results in terms of runtime.

In the era of Big Data, when we need to perform clustering on huge datasets with many data points and attributes, it is a critical issue to use a clustering technique which operates in a short amount of time, and the results of the previous chapter suggest that the proposed method could be a good choice for clustering large datasets.

BIBLIOGRAPHY

Aloise, D., A. Deshpande, P. Hansen, and P. Popat (2009). Np-hardness of euclidean sum-of-squares clustering. *Machine Learning 75*(2), 245–248.

Arthur, D. and S. Vassilvitskii (2007). k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pp. 1027–1035. Society for Industrial and Applied Mathematics.

Bache, K. and M. Lichman (2013). UCI machine learning repository.

Bahmani, B., B. Moseley, A. Vattani, R. Kumar, and S. Vassilvitskii (2012). Scalable k-means++. *Proceedings of the VLDB Endowment 5*(7), 622–633.

Bock, H.-H. (2007). Clustering methods: a history of k-means algorithms. In *Selected contributions in data analysis and classification*, pp. 161–172. Springer.

Bottou, L. and Y. Bengio (1995). Convergence properties of the k-means algorithms. In *Advances in Neural Information Processing Systems 7*. Citeseer.

Celebi, M. E. (2011). Improving the performance of k-means for color quantization. *Image and Vision Computing 29*(4), 260–271.

Celebi, M. E., H. A. Kingravi, and P. A. Vela (2013). A comparative study of efficient initialization methods for the k-means clustering algorithm. *Expert Systems with Applications 40*(1), 200–210.

Chitade, A. Z. and S. Katiyar (2010). Colour based image segmentation using k-means clustering. *International Journal of Engineering Science and Technology 2*(10), 5319–5325.

Coates, A., A. Y. Ng, and H. Lee (2011). An analysis of single-layer networks in unsupervised feature learning. In *International Conference on Artificial Intelligence and Statistics*, pp. 215–223.

Dasgupta, S. (2000). Experiments with random projection. In *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*, pp. 143–151. Morgan Kaufmann Publishers Inc.

Forbes, C., M. Evans, N. Hastings, and B. Peacock (2011). *Statistical distributions*. John Wiley & Sons.

Graham, K. and D. E. Knuth (1989). Patashnik, concrete mathematics. In *A Foundation for Computer Science*.

Hamerly, G. and C. Elkan (2002). Alternatives to the k-means algorithm that find better clusterings. In *Proceedings of the eleventh international conference on Information and knowledge management*, pp. 600–607. ACM.

Hochbaum, D. S. and D. B. Shmoys (1985). A best possible heuristic for the k-center problem. *Mathematics of operations research 10*(2), 180–184.

Jain, A. K. (2010). Data clustering: 50 years beyond k-means. *Pattern Recognition Letters 31*(8), 651–666.

Kaufman, L. and P. J. Rousseeuw (2009). *Finding groups in data: an introduction to cluster analysis*, Volume 344. John Wiley & Sons.

Kenney, J. F. and E. S. Keeping (1954). Mathematics of statistics-part one.

Linde, Y., A. Buzo, and R. M. Gray (1980). An algorithm for vector quantizer design. *Communications, IEEE Transactions on 28*(1), 84–95.

Lloyd, S. (1982). Least squares quantization in pcm. *Information Theory, IEEE Transactions on 28*(2), 129–137.

Mahajan, M., P. Nimbhorkar, and K. Varadarajan (2009). The planar k-means problem is np-hard. In *WALCOM: Algorithms and Computation*, pp. 274–285. Springer.

Murthy, V., E. Vamsidhar, J. S. Kumar, and P. S. Rao (2010). Content based image retrieval using hierarchical and k-means clustering techniques. *International Journal of Engineering Science and Technology 2*(3), 209–212.

Pena, J. M., J. A. Lozano, and P. Larranaga (1999). An empirical comparison of four initialization methods for the¡ i¿ k¡/i¿-means algorithm. *Pattern recognition letters 20*(10), 1027–1040.

Redmond, S. J. and C. Heneghan (2007). A method for initialising the k-means clustering algorithm using kd-trees. *Pattern Recognition Letters 28*(8), 965–973.

Tarsitano, A. (2003). A computational study of several relocation methods for k-means algorithms. *Pattern recognition 36*(12), 2955–2966.

Williams, G., S. Hawkins, L. Gu, R. Baxter, and H. He (2002). A comparative study of rnn for outlier detection in data mining. In *Data Mining, IEEE International Conference on*, pp. 709–709. IEEE Computer Society.